Division Strategies - Strategic Trials

Compiled by: Theodore M. Savich

March 31, 2025

This is a sharing division strategy. With sharing division problems, the number of items in each group is unknown, while the number of groups and the total number of items are both known.

$\boxed{\text{Number of groups}} \times \boxed{\text{Unknown Number of items in each group}} = \boxed{\text{Total number of items}}$

Transcript

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

- **Teacher:** Mrs. Carpenter made 56 cupcakes for a birthday party. She has eight boxes to carry the cupcakes to his party. How many cupcakes should she put in each box if she wants to put the same number of cupcakes in each box?
- **Student:** [inaudible] Put seven in. Seven.
- **Teacher:** I can tell just tell you did that. Thank you very much, Victoria.

This strategy is more sophisticated than Dealing by Ones because it involves selecting an initial, reasonable group size, testing it, and then logically refining that choice as needed.

Description of Strategic Trials:
Begin with an initial trial number for the items per group. **Utilize a multiplication strategy** to calculate the total number of items and verify it against the given total. Adjust your trial number upward or downward as necessary, and recalculate until you arrive at the correct result.

Notation and Visual Representations for Strategic Trials: Use clear notation and diagrams to illustrate the equal groups multiplication strategy you have chosen.

For example, second-grade student Victoria was tasked with determining how many cupcakes should be placed in each of 8 boxes, given a total of 56 cupcakes. She initially assumed 8 cupcakes per box and employed a doubling method to compute the total:

$$8 + 8 = 16$$

$$16 + 16 = 32$$

$$32 + 32 = 64$$

Seeing that 64 exceeded the given total, she then tried 6 cupcakes per box:

$$6 + 6 = 12$$

$$12 + 12 = 24$$

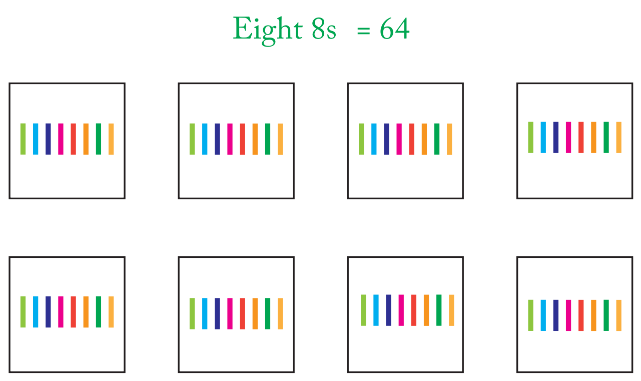
$$24 + 24 = 48$$

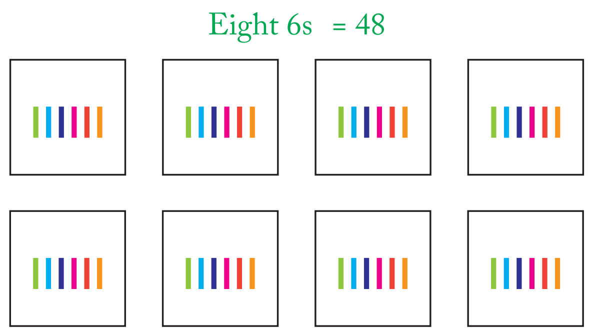
Realizing 48 was too low, Victoria understood she was estimating the number of cupcakes per box. After trying 8 (which was too high) and 6 (which was too low), she decided to test 7 cupcakes per box:

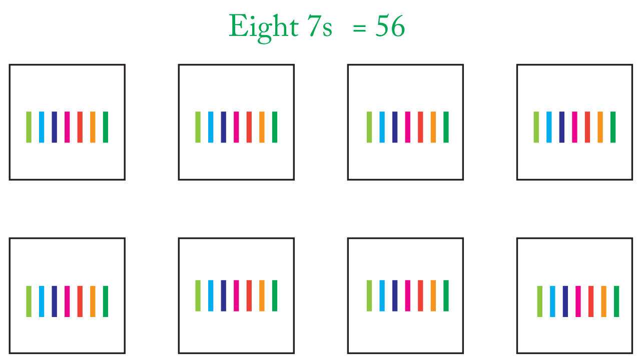
$$7 + 7 = 14$$

$$14 + 14 = 28$$

$$28 + 28 = 56 \quad \text{(using her addition strategy)}$$







She concluded that each box should contain 7 cupcakes. In class, we highlighted that her method was not merely "trial and error," but a thoughtful process of strategic adjustment. When the initial guess was too high, she adjusted downward, and when it was too low, she adjusted upward. This iterative process is a hallmark of strategic trials.

Strategic Trials

Strategy Overview

Strategic Trials involves testing different grouping configurations to find the correct division outcome. This strategy is iterative and relies on trial-and-error to determine the appropriate number of groups or the group size required for division.

Automaton Design

We design a **Pushdown Automaton (PDA)** that systematically:

1. Attempts a trial grouping by pushing a trial marker T and assigning a set of elements.
2. Checks whether the trial group meets the required size.
3. Adjusts the trial group if the size is incorrect.
4. Upon a correct trial, confirms the group by pushing a group identifier G and then outputs the final grouping.

Automaton Tuple

The PDA is defined as the 7-tuple

$$M = (Q,\, \Sigma,\, \Gamma,\, \delta,\, q_{0/accept},\, \#,\, F)$$

where:

- $Q = \{q_{0/accept}, q_{\text{trial}}, q_{\text{check}}, q_{\text{adjust}}, q_{\text{output}}\}$ is the set of states. (Here, $q_{0/accept}$ serves as both the start and the accepting state.)
- $\Sigma = \{E\}$ is the input alphabet (with E representing an element).
- $\Gamma = \{\#, T, G\}$ is the stack alphabet:
  - $\#$ is the bottom-of-stack marker.
  - T represents a trial grouping.
  - G represents a confirmed group.
- $q_{0/accept}$ is the start (and accept) state.
- $\#$ is the initial stack symbol.
- $F = \{q_{0/accept}\}$ is the set of accepting states.

State Transition Table

Current State	Input Symbol	Stack Top	Next State	Stack Operation	Description
$q_{0/accept}$	ε	—	q_{trial}	Push $\#$	Initialize
q_{trial}	ε	any	q_{check}	Push T; assign a trial group	Attempt trial
q_{check}	ε	any	q_{output}	(If trial correct: push G)	Trial correct
q_{check}	ε	any	q_{adjust}	—	Trial incorrect
q_{adjust}	ε	any	q_{trial}	Adjust trial	Modify trial group
q_{output}	ε	any	$q_{0/accept}$	Count G's	Output final grouping

Automaton Behavior

1. **Initialization:**
   - Start in $q_{0/accept}$, push $\#$ onto the stack.
   - Transition to q_{trial} to begin the trial process.
2. **Attempting a Trial:**

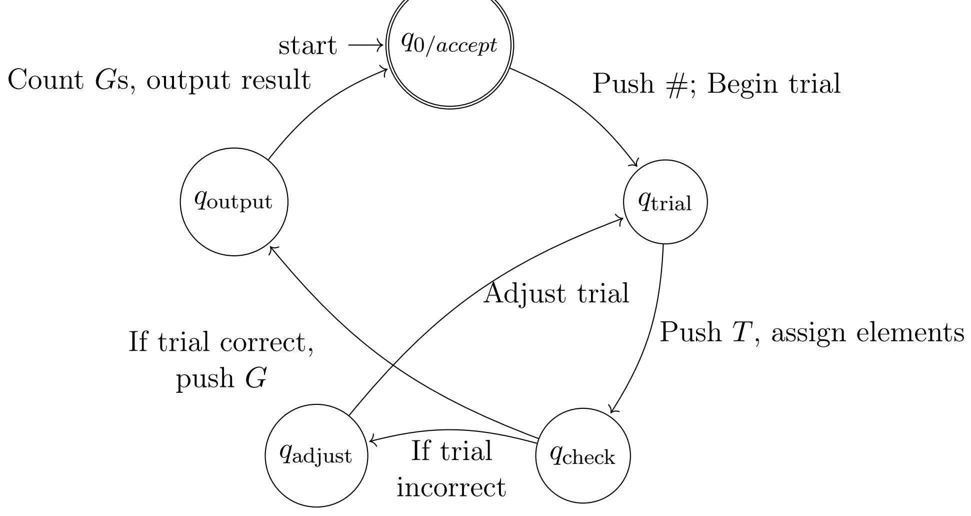
- In q_{trial}, push T to represent a trial group and assign a set of elements to it.
- Transition to q_{check}.

3. **Checking the Trial:**
   - In q_{check}, evaluate if the trial group meets the required size.
   - If the trial is correct, push a confirmed group G and transition to q_{output}.
   - If the trial is incorrect, transition to q_{adjust}.

4. **Adjusting the Trial:**
   - In q_{adjust}, modify the trial group size (by adding or removing elements).
   - Return to q_{trial} to try again.

5. **Outputting the Result:**
   - In q_{output}, count the number of confirmed groups (G symbols) on the stack.
   - Output the final grouping and transition back to $q_{0/accept}$ (the merged start/accept state).

Circular PDA Diagram



Example Execution

Problem: Divide 24 items into groups of 8 using strategic trials.

1. **Start:**
   - The initial stack contains: # followed by 24 E symbols.

2. **Trial 1:**
   - In q_{trial}, a trial group of 7 elements is attempted (push T, assign 7 E symbols).
   - In q_{check}, the trial is evaluated: $7 \neq 8$, so transition to q_{adjust}.

3. **Adjust Trial:**

- In q_{adjust}, the trial is modified (e.g., increase group size to 8).
- Return to q_{trial} for a new attempt.

4. **Trial 2:**
    - In q_{trial}, attempt a trial group of 8 elements.
    - In q_{check}, the trial is correct $(8 = 8)$; a confirmed group G is pushed.

5. **Repeat:**
    - Continue trials until all 24 items are grouped.
    - Final output: 3 groups of 8.

Iterative Handling of Trials

The PDA iteratively attempts different group sizes, adjusting the trial configuration as needed based on feedback from the check phase. This iterative process continues until the correct grouping is achieved, ensuring an accurate division.

HTML Implementation

```html
<!DOCTYPE html>
<html>
<head>
    <title>Division: Strategic Trials</title>
    <style>
        body { font-family: sans-serif; }
        .container { max-width: 800px; margin: 10px auto; padding: 10px;}
        .control-section, .trials-section, .result-section {
            margin-bottom: 20px; padding: 10px; border: 1px solid #eee;
            background-color: #f9f9f9; border-radius: 5px;
        }
        label { margin-right: 5px;}
        input[type=number] { width: 60px; margin-right: 15px;}
        button { padding: 5px 10px; font-size: 1em; margin-right: 5px; }
        #statusMessage { color: #e65c00; font-weight: bold; margin-left: 15px;}

        .trial-visualization {
           margin-top: 15px;
           padding-top: 10px;
           border-top: 1px dashed #ccc;
        }
         .group-container { /* Container for all groups in a trial */
            display: flex;
            flex-wrap: wrap; /* Allow groups to wrap */
            gap: 10px; /* Space between groups */
            margin-bottom: 5px;
         }
        .group-box {
            display: inline-block; /* Display groups inline */
            border: 1px solid #999;
            padding: 4px;
            background-color: #e8f4ff;
            min-width: 40px; /* Minimum width */
            text-align: center; /* Center items */
         }
         .group-box-label { font-size: 0.8em; color: #555; margin-bottom: 3px; display:
            block;}
        .item-block {
           display: inline-block; /* Items side-by-side */
           width: 8px; height: 8px; margin: 1px; /* Smaller items */
           background-color: #6495ED; /* Cornflower blue */
           border: 1px solid #444;
        }
        .trial-summary { font-weight: bold; margin-top: 5px; }
        .trial-correct { color: darkgreen; }
        .trial-incorrect { color: darkred; }
        #finalResultValue { font-size: 1.5em; font-weight: bold; color: darkgreen; }
    </style>
</head>
<body>
<div class="container">

```

```
    <h1>Division Strategies - Strategic Trials</h1>

    <div class="control-section">
       <label for="stratTotalInput">Total Items:</label>
       <input type="number" id="stratTotalInput" value="56" min="1"> <!-- Example -->
       <label for="stratGroupsInput">Number of Groups:</label>
       <input type="number" id="stratGroupsInput" value="8" min="1"> <!-- Example -->
       <button onclick="setupTrialSimulation()">Set Up / Reset</button>
       <button onclick="performNextTrial()" id="trialBtn" disabled>Perform Next Trial</
          button>
        <span id="statusMessage"></span>
    </div>

    <div class="trials-section">
        <strong>Trials:</strong>
        <div id="trialsDisplay">
            <!-- Trial visualizations will be added here -->
        </div>
    </div>

     <div class="result-section">
       <strong>Result (Items per group):</strong> <span id="finalResultValue">?</span>
    </div>


    <script>
       // --- Simulation State Variables ---
       let totalItems = 0;
       let numGroups = 0;
       let currentTrialSize = -1; // -1 indicates simulation not started or needs initial
           guess
       let attempts = []; // Stores history: { trialSize: number, trialResult: number,
          outcome: string }
       let finalGroupSize = null; // The correct answer when found
       let isTrialComplete = true;

       // --- DOM Element References ---
       const totalInput = document.getElementById("stratTotalInput");
       const groupsInput = document.getElementById("stratGroupsInput");
       const finalResultValueSpan = document.getElementById("finalResultValue");
       const trialsDisplay = document.getElementById("trialsDisplay");
       const trialBtn = document.getElementById("trialBtn");
       const statusMessage = document.getElementById("statusMessage");

       // --- Simulation Functions ---
       function setupTrialSimulation() {
          totalItems = parseInt(totalInput.value);
          numGroups = parseInt(groupsInput.value);

          if (isNaN(totalItems) || isNaN(numGroups) || numGroups <= 0 || totalItems < 0)
              {
             statusMessage.textContent = "Please␣enter␣valid␣positive␣numbers␣(Groups␣>␣
                0).";
             trialBtn.disabled = true;
```

```
                isTrialComplete = true;
                finalResultValueSpan.textContent = "?";
                trialsDisplay.innerHTML = ""; // Clear previous trials
                return;
            }

            // Make the first guess intentionally off (e.g., +/- 1 or 2 from rough estimate)
            let roughEstimate = Math.max(1, Math.round(totalItems / numGroups)); // Ensure guess is at least 1
            let randomOffset = Math.random() < 0.5 ? (roughEstimate > 1 ? -1 : 1) : 1; // Offset by +/- 1
            currentTrialSize = roughEstimate + randomOffset;
            // Ensure guess isn't accidentally correct if estimate was close
            if (currentTrialSize * numGroups === totalItems && currentTrialSize > 1) {
                currentTrialSize--; // Adjust if first guess happens to be right
            }
             if (currentTrialSize <= 0) currentTrialSize = 1; // Ensure guess is at least 1


            attempts = []; // Clear history
            finalGroupSize = null;
            isTrialComplete = false;

            statusMessage.textContent = `Ready. Initial trial guess: ${currentTrialSize} items per group.`;
            finalResultValueSpan.textContent = "?";
            trialsDisplay.innerHTML = ""; // Clear previous trials visually
            trialBtn.disabled = false;
        }

        function performNextTrial() {
            if (isTrialComplete) {
                statusMessage.textContent = "Found correct group size! Press Reset to start again.";
                trialBtn.disabled = true;
                return;
            }

            statusMessage.textContent = `Trying ${currentTrialSize} items per group...`;

            // 1. Multiply to get trial total
            const trialResult = currentTrialSize * numGroups;

            // 2. Check against actual total
            let outcome = "";
            let outcomeClass = "";
            if (trialResult === totalItems) {
                outcome = "Correct!";
                outcomeClass = "trial-correct";
                finalGroupSize = currentTrialSize;
                isTrialComplete = true;
                trialBtn.disabled = true; // Disable button once correct
```

```
                 statusMessage.textContent = `Found correct group size: ${finalGroupSize}!`;
                 finalResultValueSpan.textContent = finalGroupSize;
            } else if (trialResult < totalItems) {
                outcome = `Too Low (${trialResult} < ${totalItems})`;
                outcomeClass = "trial-incorrect";
            } else { // trialResult > totalItems
                outcome = `Too High (${trialResult} > ${totalItems})`;
                 outcomeClass = "trial-incorrect";
            }

            // 3. Store attempt
            attempts.push({
                trialSize: currentTrialSize,
                trialResult: trialResult,
                outcome: outcome,
                outcomeClass: outcomeClass
            });

             // 4. Draw this attempt
             drawTrialVisualization(currentTrialSize, numGroups, trialResult, outcome, outcomeClass);


            // 5. Adjust for next trial (if not correct)
            if (!isTrialComplete) {
                if (trialResult < totalItems) {
                     // Increase guess (could be smarter, e.g., based on how far off)
                     currentTrialSize++;
                } else {
                     // Decrease guess
                     currentTrialSize--;
                     if (currentTrialSize <= 0) currentTrialSize = 1; // Don't guess 0 or negative
                }
                 statusMessage.textContent += ` Adjusting guess to ${currentTrialSize}.`;
            }
        }

        function drawTrialVisualization(trialSize, groups, result, outcome, outcomeClass) {
            const trialDiv = document.createElement('div');
            trialDiv.className = 'trial-visualization';

            const groupContainer = document.createElement('div');
            groupContainer.className = 'group-container';

            for (let g = 0; g < groups; g++) {
                const groupBox = document.createElement("div");
                groupBox.className = "group-box";
                // groupBox.innerHTML = `<span class="group-box-label">Group ${g + 1}</span>`; // Optional label

                 // Arrange items within the box (e.g., simple horizontal flow)
                 let itemsHtml = '';
                 let itemsPerRow = Math.max(5, Math.ceil(Math.sqrt(trialSize))); // Simple layout heuristic
                 for(let i = 0; i < trialSize; i++) {
                     itemsHtml += `<span class="item-block"></span>`;
                     if ((i + 1) % itemsPerRow === 0) itemsHtml += '<br>'; // Add line break
                 }
                 groupBox.innerHTML += itemsHtml;
                 groupContainer.appendChild(groupBox);
            }
            trialDiv.appendChild(groupContainer);

             const summary = document.createElement('div');
             summary.className = 'trial-summary';
             summary.innerHTML = `Trial: ${groups} groups  ${trialSize} items/group = ${result}. <span class="${outcomeClass}">${outcome}</span>`;
             trialDiv.appendChild(summary);


            trialsDisplay.appendChild(trialDiv);
            trialsDisplay.scrollTop = trialsDisplay.scrollHeight; // Scroll to bottom
        }


        // --- Helper SVG/Typeset Functions (Not needed for this block viz) ---
        function typesetMath() { /* Placeholder */ }

        // --- Initialize ---
        setupTrialSimulation(); // Initialize state on load


</script>

</div> <!-- End Container -->
</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].