The course staff spent several hours grading your midterms on Saturday morning and afternoon, so I’m happy to report they’ve been graded, and your graded midterms will be published via Gradescope ahead of today’s lecture. The exam was intended to be challenging, but many of you did brilliantly, and most of you did well enough that I’m happy to go with a traditional curve for an accelerated course, where I set the median grade to sit at the A-/B+ border.

The complete histogram of grades is presented below, where each dot represents a single exam score (out of 70 points).

You can determine your letter grade by looking up your score in the following table:

<table>
<thead>
<tr>
<th>Range</th>
<th>Grade</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>65–70</td>
<td>A+</td>
<td>3</td>
</tr>
<tr>
<td>50–64</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>43–49</td>
<td>A–</td>
<td>8</td>
</tr>
<tr>
<td>37–42</td>
<td>B+</td>
<td>3</td>
</tr>
<tr>
<td>28–36</td>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>20–27</td>
<td>B–</td>
<td>3</td>
</tr>
<tr>
<td>16–19</td>
<td>C+</td>
<td>3</td>
</tr>
<tr>
<td>12–15</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>10–11</td>
<td>C–</td>
<td>0</td>
</tr>
<tr>
<td>00–09</td>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

Median = 43 (61.4%)
Solution 1: Simple JavaScript expressions, statements, and functions (10 points)

(1a) \[ 8 \% 3 + 19 \% 11 \% 5 \]

\[ "c" < "cc" \&\& "45" < "5" \]

\[ 20 + "" + 8 / 4 + 36 / 12 \]

(1b) 1

(1c) "eswanest"

Solution 2: Using graphics and animation (15 points)

```javascript
/*
 * Function: DrawPumpkinPies
 * -------------------------
 * Draws the first of potentially several pumpkin pies
 * in the screen, and then installs the mouse event
 * handling needed to consume a slice of pie when clicked.
 */
function DrawPumpkinPies() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let pieRadius = INITIAL_PIE_RADIUS;
    let numRemaining = 0;

    /*
     * Function: refill
     * ---------------
     * Draws a pumpkin pie, and initializes the
     * state needed to keep track of how many slices
     * remain and how large the new pie should be.
     */
    let refill = function() {
        numRemaining = NUM_PIECES;
        drawPie(gw, pieRadius);
        pieRadius += PIE_RADIUS_DELTA;
    };

    /*
     * Function: clickAction
     * ---------------------
     * Determines whether or not a pie slice occupies
     * the location of the mouse event, and if so,
     * removes it and updates the bookkeeping needed to
     * determine whether all slices have been consumed.
     * If so, a new pie is delivered after a one-second delay.
     */
    let clickAction = function(e) {
        let slice = gw.getElementAt(e.getX(), e.getY());
        if (slice === null) return;
        gw.remove(slice);
        numRemaining--;
        if (numRemaining == 0) setTimeout(refill, ONE_SECOND);
    }
};
```
```javascript

// Function: drawPie
function drawPie(gw, pieRadius) {
    let ulx = gw.getWidth()/2 - pieRadius;
    let uly = gw.getHeight()/2 - pieRadius;
    let sweep = 360 / NUM_PIECES;
    for (let i = 0; i < NUM_PIECES; i++) {
        let slice = GArc(ulx, uly, 2 * pieRadius, 2 * pieRadius,
                         i * sweep, sweep);
        slice.setColor("Black");
        slice.setFilled(true);
        slice.setFillColor("Orange");
        gw.add(slice);
    }
}

// Function: advanceRotorSetting
function advanceRotorSetting(setting) {
    let next = ",";
    let carry = true;
    let base = "A".charCodeAt(0);
    for (let i = setting.length - 1; i >= 0; i--) {
        let ch = setting.charAt(i);
        if (carry) {
            let offset = ch.charCodeAt(0) - base;
            offset++;
            offset %= 26;
            ch = String.fromCharCode(base + offset);
            carry = offset === 0;
        }
        next = ch + next;
    }
    return next;
}
```

**Solution 3: Strings (15 points)**

```javascript

// Function: advanceRotorSetting
function advanceRotorSetting(setting) {
    let next = ",";
    let carry = true;
    let base = "A".charCodeAt(0);
    for (let i = setting.length - 1; i >= 0; i--) {
        let ch = setting.charAt(i);
        if (carry) {
            let offset = ch.charCodeAt(0) - base;
            offset++;
            offset %= 26;
            ch = String.fromCharCode(base + offset);
            carry = offset === 0;
        }
        next = ch + next;
    }
    return next;
}
```
Solution 4: Arrays (15 points)

```javascript
/*
 * Function: merge
 * ---------------
 * Traverses the array of intervals, sorted by start-time, and merges
 * overlapping intervals into one.
 */
function merge(intervals) {
    let i = 0;
    while (i < intervals.length - 1) {
        let curr = intervals[i];
        let next = intervals[i + 1];
        if (curr[1] >= next[0]) {
            curr[1] = Math.max(curr[1], next[1]);
            intervals.splice(i + 1, 1);
        } else {
            i++;
        }
    }
}
```

Solution 5: Working with data structures (15 points)

```javascript
/*
 * Function: invert
 * -------------
 * Accepts a url-to-term-sequence map, builds the inverted map,
 * and returns it.
 */
function invert(urlToTerms) {
    let inverted = {};
    for (let url in urlToTerms) {
        let terms = urlToTerms[url];
        for (let i = 0; i < terms.length; i++) {
            let term = terms[i];
            if (inverted[term] === undefined) inverted[term] = {};
            if (inverted[term][url] === undefined) inverted[term][url] = 0;
            inverted[term][url]++;
        }
    }
    return inverted;
}
```