Your trusty CS106AJ staff cranked through two grading sessions last Thursday and Friday to press through your final exams, and I’m happy to report they’re graded, and that final grades have even been computed as submitted to access. The median on the final exam was a 69 out of 100, and grades ranged from 12 all the way up to a 94.

The complete histogram of grades is presented below, where each vertical bar represents a single exam score; lower scores are on the left and higher scores are on the right.

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>90–100</td>
<td>A+</td>
<td>3</td>
</tr>
<tr>
<td>81–89</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>72–80</td>
<td>A−</td>
<td>6</td>
</tr>
<tr>
<td>65–71</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>53–64</td>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>50–52</td>
<td>B−</td>
<td>1</td>
</tr>
<tr>
<td>45–49</td>
<td>C+</td>
<td>1</td>
</tr>
<tr>
<td>37–44</td>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>34–36</td>
<td>C−</td>
<td>1</td>
</tr>
<tr>
<td>19–33</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>00–18</td>
<td>NP</td>
<td>1</td>
</tr>
</tbody>
</table>

Median = 69  
Mean = 65.3
Solution 1—Short answer (10 points)

1a) The code provided, batty variable names notwithstanding, is an implementation of Kadane’s algorithm, which uses a technique called dynamic programming to compute the largest subarray sum in an array of integers.

\[ \text{perplexity}([-2, 1, -3, 4, -1, 2, 1, -5, 7, -10]) ; \]

produces the following

| -2 | 1 | 1 | 4 | 4 | 5 | 6 | 6 | 8 | 8 |

1b) **Answer to Problem 1b: 312056**
Solution 2—Simple graphics (15 points)

```javascript
const LIKERT_CHART_WIDTH = 200;
const LIKERT_CHART_HEIGHT = 20;
const LIKERT_CHART_BORDER = "Gray";
const LIKERT_CHART_COLORS = ["Red", "Orange", "LightGray", "Green", "Blue"];

/**
 * Function: createLikertChart
 * ---------------------------
 * Assuming the supplied responses array is of length 5,
 * and that the sum of the five numbers is 100, creates
 * and returns a Likert chart as described in the above problem
 * statement.
 */
function createLikertChart(percentages) {
    let bar = GCompound();
    let inset = percentages[2]/2 + percentages[3] + percentages[4];

    for (let i = 0; i < 5; i++) {
        if (percentages[i] > 0) {
            let box = GRect(inset, 0, percentages[i], LIKERT_CHART_HEIGHT);
            box.setColor(LIKERT_CHART_COLORS[i]);
            box.setFill(true);
            bar.add(box);
            inset += percentages[i];
        }
    }

    let bounds = GRect(0, 0, LIKERT_CHART_WIDTH, LIKERT_CHART_HEIGHT);
    bounds.setColor(LIKERT_CHART_BORDER);
    bar.add(bounds);
    return bar;
}
```
Solution 3—Interactive graphics (20 points)

/* Constants */
const GWINDOW_WIDTH = 700;
const GWINDOW_HEIGHT = 400;
const CAT_RADIUS = 10;
const CAT_COLOR = "Silver";
const DELAY = 4;

/* Derived Constants */
const CAT_DIAMETER = 2 * CAT_RADIUS;

/**
 * Function: createCat
 * ---------------
 * Creates a small, silver circle to represent the cat, places it
 * at the center of the graphics window, and then returns it.
 * */
function createCat(gw) {
  let cat = GOval(GWINDOW_WIDTH/2 - CAT_RADIUS, GWINDOW_HEIGHT/2 - CAT_RADIUS,
                  CAT_DIAMETER, CAT_DIAMETER);
  cat.setFilled(true);
  cat.setFillColor(CAT_COLOR);
  gw.add(cat);
  return cat;
}

/**
 * Function: CatAndMouse
 * ---------------------
 * Simulates a cat and mouse game where the cat (represented
 * as a small, silver circle with a black border) advances
 * towards the mouse (the location defined by the actual location
 * of the ever moving mouse), 1 px every 4 milliseconds. When
 * the cat actually catches the mouse, the mouse dies and
 * goes to cat heaven (i.e. it's removed from the graphics window),
 * the animation timer is removed, and the program effectively ends.
 * */
function CatAndMouse() {
  let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
  let cat = createCat(gw);

  let started = false;
  let lastmx, lastmy;

  gw.addEventListener("mousemove", function (e) {
    lastmx = e.getX();
    lastmy = e.getY();
  });
window.addEventListener("click", function (e) {
  if (started) return;
  lastmx = e.clientX; // just in case the mouse hasn't moved yet
  lastmy = e.clientY; // (don't require this from the students, though)
  let timer = setInterval(function() {
    if (gw.getElementAt(lastmx, lastmy) !== null) {
      clearInterval(timer);
      gw.remove(cat);
    } else {
      let dx = lastmx - (cat.getX() + CAT_RADIUS);
      let dy = lastmy - (cat.getY() + CAT_RADIUS);
      let distance = Math.sqrt(dx * dx + dy * dy);
      cat.move(dx/distance, dy/distance);
    }
  }, DELAY);
  started = true;
});
Solution 4—Strings (15 points)

/**
 * Function: recover
 * ---------------
 * Accepts an array of integers produced as described in the problem
 * statement, and returns the string they encode.
 */
function recover(encoding) {
    let str = "";
    let legend = ALPHABET;
    for (let i = 0; i < encoding.length; i++) {
        let index = encoding[i];
        str += legend.charAt(index);
        legend = legend.charAt(index) +
        legend.substr(0, index) + legend.substr(index + 1);
    }
    return str;
}
Solution 5—Arrays (10 points)

/**
 * Function: dedupe
 * ------------
 * Updates the supplied array such that all duplicates
 * are removed. The implementation is designed to work
 * for arrays of any single primitive type (e.g. an array
 * of numbers, or an array of strings, or an array of bools)
 */
function dedupe(array) {
    for (let i = array.length - 1; i >= 0; i--) {
        if (array.indexOf(array[i]) < i) {
            array.splice(i, 1);
        }
    }
} // note that there are many other acceptable solutions to this
Solution 6—Working with data structures (15 points)

/**
 * Predicate Function: mappingIsValid
 * ----------------------------------
 * Returns true if the supplied gene is a valid encoding
 * of the supplied amino acid sequence, and false otherwise.
 */
function mappingIsValid(gene, sequence) {
    if (gene.length !== 3 * (sequence.length + 2)) return false;
    let start = gene.substring(0, 3);
    if (start !== START_CODON) return false;
    gene = gene.substring(3);
    for (let i = 0; i < sequence.length; i++) {
        let codons = MAPPINGS[sequence[i]];
        let codon = gene.substring(0, 3);
        if (codons.indexOf(codon) === -1) return false;
        gene = gene.substring(3);
    }
    let stop = gene;
    return STOP_CODONS.indexOf(stop) !== -1;
}
Solution 7—Reading data structures from embedded XML (15 points)

/**
 * Factory Function: PresidentialWordCloud
 * ---------------------------------------
 * Scrapes the XML as per the problem specification and
 * returns an object with two methods that access the data
 * in a way that's also detailed in the problem statement.
 */
function PresidentialWordCloud() {
  let wordsMap = {};
  let tagsMap = {};
  let cloudXML = document.getElementById("CloudData");
  processCloudXML(cloudXML, wordsMap, tagsMap);
  let getAllWords = function(title, date) {
    let result = wordsMap[title + ":" + date];
    return (result === undefined) ? [] : result;
  };
  let getAllTags = function(title, date, weight) {
    let result = tagsMap[title + ":" + date];
    if (result === undefined) return [];
    result = result["" + weight];
    return (result === undefined) ? [] : result;
  };
  return {
    getAllWords: getAllWords,
    getAllTags: getAllTags
  };
}

/**
 * Function: processCloudXML
 * ------------------------
 * Crawls over the supplied cloudXML data and populates
 * the wordsMap and tagsMap data structures.
 */
function processCloudXML(cloudXML, wordsMap, tagsMap) {
  let speeches = cloudXML.getElementsByTagName("speech");
  for (let i = 0; i < speeches.length; i++) {
    let title = speeches[i].getAttribute("title");
    let date = speeches[i].getAttribute("date");
    let tags = speeches[i].getElementsByTagName("tag");
    let key = title + ":" + date;
    wordsMap[key] = [];
    tagsMap[key] = {};
    for (let j = 0; j < tags.length; j++) {
      let word = tags[j].getAttribute("word");
      let color = tags[j].getAttribute("color");
      let weight = tags[j].getAttribute("weight");
      wordsMap[key].push(word);
      if (tagsMap[key][weight] === undefined) tagsMap[key][weight] = [];
      tagsMap[key][weight].push({word: word, color: color});
    }
  }
}