Information

Students took the exam. Scores ranged from 1 to 20, with a median of 10 and an average of 10.6. There were 42 scores between 15 and 20, 82 between 11 and 15.5, 80 between 6 and 10.5, and 42 between 1 and 5.5. (Were you to receive a grade of 11 on a midterm exam, 48 on the final exam, plus good grades on homework and quizzes, you would receive an A—; similarly, a test grade of 11 may be projected to a B—.) There were four versions of the exam, A, B, C, and D. (The version indicator appears at the bottom of the first page.) Versions A and C were identical except for the order of the problems. Versions B and D were also identical except for the order of the problems.

If you think we made a mistake in grading your exam, describe the mistake in writing, and include the description of the exam to your lab T.A. or to Mike Clancy. We read the entire exam.

Grading standards for versions A and B

Problem 1 (1 point)

Some credit on a problem and did not put your name on the page, you lost ½ point. If you did not indicate your lab section and T.A., you lost ½ point. If you put the names of your neighbors on the front page of a test, you lost ½ point.

Problem 2 (3 points on version A, 4 points on version B)

Version A: This problem involved analysis of a hash function function for use with a table of size 10000 storing intervals whose endpoints were between -100 and 100.

```java
public int hashCode() {
    return left + right;
}
```

Version B: Hash does not spread out collisions evenly. 201 nonempty intervals hash to 0; intervals hash to composite table positions; only two intervals hash to each number between -100 and 100; no intervals hash to prime numbers between 10000 or to their negative counterparts. (14,183 values between -10,000 and 10,000 cannot be returned by the hash function.)

Grades for this problem were awarded as follows:

- For saying that the function was bad;
- For noting either that composite table positions had a lot of collisions or that prime table positions had few;
- For noting both the above, or for noting one of them and making it clear there were table cells that have a lot of collisions as well as cells that have

<table>
<thead>
<tr>
<th>word</th>
<th>maximum length</th>
<th>maximum(hashCode value)</th>
<th>maximum(hashCode value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1338</td>
<td>1768</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>28910</td>
<td>3660</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6014</td>
<td>7654</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12222</td>
<td>15372</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21638</td>
<td>30988</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>49470</td>
<td>62220</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>99134</td>
<td>124684</td>
<td></td>
</tr>
</tbody>
</table>

(Hardly anyone gave this answer.)

This problem was worth 4 points. We awarded 1 point for each correct size and 1 point for a correct corresponding reason, except that we gave 2 points for two sets of table sizes for which the reason was the same (divisible by 10 and table sizes divisible by 4). We also gave 2 points for “small table sizes because they produce a lot of collisions”. The answers were claimed because they waste a lot of space” only received 1 point unless space was explained in terms of uneven distribution of words.
Problem 2 (8 points on version A, 7 points on version B)

The problem was derived from work you did in lab assignment 4, homework assignment 4 (version A), lab assignment 5, homework 6, and project 2 (version B). This problem 3 on versions C and D.

Section A involved analyzing a Vector version of the interval combining function.

```java
public void combine() {
    for (int k=0; k < myIntervals.size(); k++) {
        Interval current = (Interval) myIntervals.elementAt(k);
        Interval next = (Interval) myIntervals.elementAt(k+1);
        if (!current.overlaps(next)) {
            myIntervals.addElementAt((current.extendThrough(next), k);
            myIntervals.removeElementAt(k+1);
        }
    }
}
```

Section B involved analyzing a Vector version of the function that deletes squares.

```java
public void deleteAll(int x, int y) {
    for (int k=0; k < mySquares.size(); k++) {
        if (((Square) mySquares.elementAt(k)).contains(x, y)) {
            mySquares.removeElementAt(k);
        }
    }
}
```

The functions had a bug resulting from the rearrangement of index values resulting from deleting a vector element. When element k is deleted, its successor is the new element k; since the loop variable is incremented each time through the loop, an error occurring square that follows one that gets deleted is not examined. Note that the `removeElementAt` method also decreases the vector's size, something that some students apparently didn't know.

It was worth 2 points. The answer in version A was that any list with a sequence of three or more overlapping intervals is handled incorrectly; in version B, any list of consecutive squares that contain the point \((x, y)\) is handled incorrectly. 1 point was awarded for a vague but possibly correct answer, an answer that gave only an example, or an answer that explained what was wrong with the code but failed to describe the vectors for which it would perform incorrectly.

It was worth 2 points in version A and 1 point in version B. Any of the following forms received full credit:

- A statement that decrements k to the if clause;
- An else clause that increments k, and then remove the k increment from the for loop header;
- Place the for loop header so as to process the list elements in reverse order;
- Change the for to a while loop that incorporated one of the above modifications.

CS 61B (Clancy) Solutions and grading standards for exam 2
Spring 1999

A more extensive fix received only half credit (1 point on version B). No points were awarded to an incorrect fix, nor to a more complicated "deleted" elements as empty intervals or squares of size 1.

Part C involved comparing timings of the fixed code and the efficiency of the combine and `deleteAll` methods compared to the version of `intervalCombination` homework results from deleting list elements from the middle of the list to simulate the computer's execution of the code. The corrected version of each method required shifting the elements that follow. The more elements a list has, the more list elements, and therefore will take more time.

One may observe two features of the listed timings:

- One column's timings are uniformly greater than the other's, which correspond to the corrected version. Moreover, the test list makes the corrected version work harder, namely one with a list of one or three or more overlapping intervals or one with a list of many squares.
- Also, the increase in the time is greater than linear in N for roughly triple when N is increased from 512 to 1024 and from the rate of growth as N increases is itself increasing. This growth is a function of the test list that contains a lot of intervals. On lists with relatively few elements, both methods will take time proportional to N.

This part was worth 4 points, 1 for identifying which program went faster, 1 for justifying the answer, 1 for identifying the reason, and 1 for justifying that answer. A description of the test list as one containing intervals/squares" was sufficient.

A fix to the code that produced correct behavior usually also produced points in part C. One could still get 2 out of the 4 points for identifying the bug, however. Some students seemed to think that combine or deleteAll deleted a list, then to the same list with all deletable elements already removed clearly not produce the listed figures.
Solutions and grading standards for exam 2

CS 61B (Clancy)  Solutions and grading standards
Spring 1999

Large intervals, incidentally, are not specifically a cause for concern, as problem specified that only nonoverlapping intervals would be stored. Any might fill up.

Part a was worth 4 points. You received at least 1 point for indicating the intervals containing only a single element rather than entire intervals being hashed. A second point was earned for any solution that applied the hash all the integers in each interval stored. The third and fourth points were given for solutions that stored an interval (actually a reference to an interval's end) as the key and that did not limit the ability of the java.util.HashTable object to the table if it filled up. A diagram inconsistent with the accompanying text lost 1 point, except that no points were deducted for a diagram that showed integers mapped to adjacent hash table chains. (A reasonable hash function probably not map adjacent integers anywhere near each other.) Many devised inapposite hash functions in an attempt to make the exam results collide, thereby losing 2 points; we had hoped to avoid this by including a "assume for the purposes of illustration".

Part b was worth 2 points. You needed to earn at least 2 points on part b to earn any points on this part. Solutions that failed to cast the value retrieved from that failed to convert an int into an integer (or that made both errors) Solutions that searched or inserted into a chain rather than letting the method do it lost 1 point.

In part c, we attempted to evaluate your solution to part b: no solution part b meant no points on part c. Giving any expression involving "N" what N was earned 6 points out of the 2 for this part. (N could be the lower or the number of intervals in the abstract collection, or the number of stored in the actual collection.) You had to say something about the "io" or the 'maximum' number of collisions (since the problem asked for worst case behavior) for full credit. 1 point was deducted for omitting a mention in this way.

void insert (Interval intvl) {
    for (int k=intvl.left; k<intvl.right; k++) {
        myIntervals.put (new Integer (k), intvl);
    }
}

Interval intervalContaining (int x) {
    return (Interval) myIntervals.get (new Integer (x));
}

Insertion behavior results from long intervals (note that insertion can be done at the start of a chain, and thus any particular integer can be constant time). Worst-case retrieval results from lots of collisions.