(Clancy) Solutions and grading standards for exam 2
1999

nformation

dents took the exam. Scores ranged from 1 to 20, with a median of 10 and an of 10.6. There were 42 scores between 16 and 20, 82 between 11 and 15.5, een 6 and 10.5, and 42 between 1 and 5.5. (Were you to receive a grade of 16 ur midterm exams, 48 on the final exam, plus good grades on homework and would receive an A–; similarly, a test grade of 11 may be projected to a B–.)

ere four versions of the exam, A, B, C, and D. (The version indicator appears ottom of the first page.) Versions A and C were identical except for the order oblems. Versions B and D were also identical except for the order of the prob-

ink we made a mistake in grading your exam, describe the mistake in writ- hand the description with the exam to your lab t.a. or to Mike Clancy. We ade the entire exam.

ns and grading standards for versions A and B

n 0 (1 point)

rned some credit on a problem and did not put your name on the page, you oint. If you did not indicate your lab section and t.a., you lost ½ point. If you ut the names of your neighbors on the front page, you lost ½ point.

n 1 (3 points on version A, 4 points on version B)

sions of this problem were based on analysis activities you did in lab assign- nd homework assignment 7.

on A, this problem involved analysis of a hashCode function for use with a le of size 10000 storing intervals whose endpoints were between –100 and

```
blic int hashCode ( ) {
 return left * right;
```

tion does not spread out collisions evenly. 201 nonempty intervals hash to 0; ervals hash to composite table positions; only two intervals hash to each mber between –100 and 100; no intervals hash to prime numbers between 10000 or to their negative counterparts. (14,188 values between –10,000 and nnot be returned by the hashCode function.)

nts for this problem were awarded as follows:

t for saying that the function was bad;

t for noting *either* that composite table positions had a lot of collisions or rime table positions had very few;

t for noting *both* the above, or for noting one of them and making it clear here were table cells that have a lot of collisions as well as cells that have

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

Saying only that some positions got a lot of collisions was insufficie since a hash function might distribute the remaining keys evenly a cells.

The hashCode function returns negative values for intervals that c some students thought that negative return values would cause tro java.util.Hashtable objects map the complete range of integers into t table index values. No deduction was made for this error, however.

Version B also involved analyzing a hash function, this one applied

```
public int hashCode ( ) {
   int h = 0;
   for (int k=0; k<s.length(); k++) {
      h = 2 * (h + s.charAt (k));
   }
   return h;
}
```

This function returns an even integer value for every word. Thus for half the table cells will be empty. For large table sizes, the same sor noticed in homework assignment 7, exercise 4, would appear becau tion of English words, as suggested in the table below. (Few student answer.)

word length	maximum hashCode value	maximum hashCode value
3	1358	1708
4	2910	3660
5	6014	7564
6	12222	15372
7	24638	30988
8	49470	62220
9	99134	124684

(Hardly anyone gave this answer.)

This problem was worth 4 points. We awarded 1 point for each corre sizes and 1 point for a correct corresponding reason, except that we 2 points for two sets of table sizes for which the reason was the sam divisible by 10 and table sizes divisible by 4). We also gave 2 points "small table sizes because they produce a lot of collisions". The answ sizes because they waste a lot of space" only received 1 point unless space was explained in terms of uneven distribution of words.

blem 2 (8 points on version A, 7 points on version B)

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

ion A involved analyzing a Vector version of the interval combining function
homework 4:

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

ion B involved analyzing a Vector version of the function that deletes squares
project 2:

```
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);
        }
    }
}
```

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

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

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

d a statement that decrements k to the if clause;

eate an else clause that increments k, and then remove the k increment from
e for loop header;

place the for loop header so as to process the list elements in reverse order;

ange the for to a while loop that incorporated one of the above modifications.

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

Part c involved comparing timings of the fixed code and the bug
ciency of the combine and deleteAll methods compared to the ve
homework results from deleting list elements from the middle
requires shifting the felements that follow. The more elements
the code takes to execute. The corrected version of each method
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
  correspond to the corrected version. Moreover, the test list m
  makes the corrected version work harder, namely one with a
  three or more overlapping intervals or one with a lot of conse
  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 fro
  the *rate* of growth as N increases is itself increasing. This gro
  test list that contains a lot of intervals. On lists with relativ
  ments, both methods will take time proportional to N.

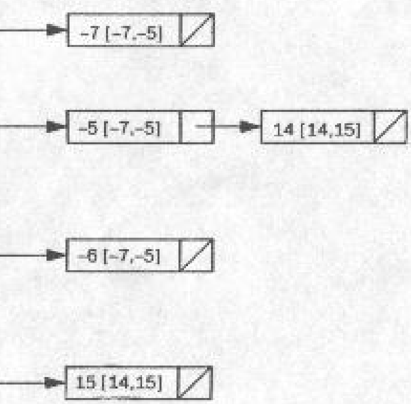
This part was worth 4 points, 1 for identifying which program v
to which column, 1 for justifying the answer, 1 for identifying th
justifying that answer. A description of the test list as one conta
table intervals/squares" was sufficient.

A fix to the code that produced linear-time behavior usually als
points in part c. One could still get 2 out of the 4 points for ident
however. Some students seemed to think that combine or delete
to a list, then to *the same list* with all deletable elements alread
clearly not produce the listed figures.

3 (8 points)

em was derived from homework assignment 4, from hashing exercises in nent 7 and homework assignment 7, and from the use you made of hash- ework assignment 8. (It was problem 2 on versions C and D.)

as to store nonoverlapping intervals in such a way that they could be uickly given any integer in the interval. That means that the interval useful as a key value; instead, the integers in the interval should be used d the interval stored multiple times in the table. Here's a diagram of how -7,–5] and [14,15] might be stored in a chained table:



n interval into the table is done as follows:

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

to retrieve an interval:

```
ic Interval intervalContaining (int x) {
eturn (Interval) myIntervals.get (new Integer (x));
```

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

Large intervals, incidentally, are not specifically a cause for concern, lem 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 intervals containing only a single element rather than entire interval hashed. A second point was earned for any solution that applied the h all the integers in each interval stored. The third and fourth points w solutions that stored an interval (actually a reference to an interval) ger key, and that did not limit the ability of the java.util.hashTable obj the table if it filled up. A diagram inconsistent with the accompanyin lost 1 point, except that no points were deducted for a diagram that sl integers mapped to adjacent hash table chains. (A reasonable hash fu probably not map adjacent integers anywhere near each other.) Many devised inappropriate hash functions in an attempt to make the exan collide, thereby losing 2 points; we had hoped to avoid this by includi "assume for the purposes of illustration".

Part b was worth 2 points. You needed to earn at least 2 points on par points on this part. Solutions that failed to cast the value retrieved fr 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 th method do it lost 1 point.

In part c, we attempted to evaluate your solution to part b; no solutio part b meant no points on part c. Giving any expression involving "N" what N was earned 0 points out of the 2 for this part. (N could be the l 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 "lo or the "maximum" number of collisions (since the problem asked for w behavior) for full credit. 1 point was deducted for omitting a mention in this way.