There are 7 problems on the exam, with 56 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a separate double-sided one-page code handout. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it’s fine if you show your work).

Put your name and loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3 &amp; 4</td>
<td>7 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 7</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>56 pts.</td>
</tr>
</tbody>
</table>
Problem 1 [5 pts. total]
Consider a hash table with chaining using an array called \texttt{hashArray}. Suppose we have a key \texttt{ourKey}, whose hash value is 27. (i.e., \texttt{hash(ourKey) returns 27}).

Part A (1). Circle the answer that best matches:

If \texttt{hashArray[27]} is not empty we know that:

i. \texttt{ourKey} is definitely present in the table

ii. \texttt{ourKey} may be present in the table

iii. \texttt{ourKey} is definitely not present in the table

Part B (1). Give an answer from the above choices (i, ii, iii) that best matches for the following:

If \texttt{hashArray[27]} is empty we know that:

Part C (2). If either of your answers to part A or B was ii, describe the other work is necessary to determine whether \texttt{ourKey} is present in the table. Do not write code, but be specific about what has to happen. (If you answered ii to both parts, give and label two answers, one for each of parts A and B; if you didn't answer ii to either, just say "no additional work necessary").

Part D (1). Suppose we have a second key, \texttt{ourKey2}, whose hash value is also 27. What do we know about the relationship between \texttt{ourKey} and \texttt{ourKey2}:

a. they have the same value

b. they are the same object (i.e., two object variables referring to the same object)

c. \texttt{ourKey} is less than \texttt{ourKey2}

d. \texttt{ourKey} is greater than \texttt{ourKey2}

e. none of the above
Problem 2 [6 pts. total]

Part A. Consider the following balanced binary search tree, whose root is shown nearest the top of the page (i.e., it's not a sideways tree):

For each of the following lookups from the tree shown above, give the sequence of keys that the target key would have to be compared with to do the lookup.

Part A. lookup Hal

Part B. lookup Bob

Part C. lookup Kat
Problem 3 [4 pts.]
Consider a Map class to store a collection of key-value pairs (no duplicate keys) with operations insert, remove, lookup, and printInOrder (the last is to print all entries in order by key). For A-D give worst case unless average case is better.

A. How long would printInOrder take in big-O terms if the Map used an unordered array representation?

B. How long would remove take if we used a balanced search tree representation?

C. How long would lookup take if we used an ordered array representation (i.e., ordered by keys)?

D. How long would insert take if we used an unordered linked list representation?

Problem 4 [3 pts.]
Show what would be in the following array after the first three passes of the insertion sort algorithm (sorting in increasing order). Hint: For an n-element array, it’s completely sorted after n-1 passes.

array before call to insertion sort:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>12</td>
<td>2</td>
<td>7</td>
<td>20</td>
<td>36</td>
<td>5</td>
<td>4</td>
<td>18</td>
<td>15</td>
</tr>
</tbody>
</table>

(array before call to insertion sort: top level shows array indices)

array after 3 passes of insertion sort:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>20</td>
<td>36</td>
<td>5</td>
<td>4</td>
<td>18</td>
<td>15</td>
</tr>
</tbody>
</table>
Problem 5 [10 points]

Suppose you want to create a Map<Student, String> that maps students to their letter grades (letter grades such as "A", or "B+" are represented as the String value in the map). One of the things we want to use this Map for is to create a class list of students and their grades in sorted order by name, so we will choose a TreeMap for this application. When we traverse the TreeMap, the results will come out in the order shown in this example (note that for students with the same name, they are be ordered by student ID):

<table>
<thead>
<tr>
<th>last</th>
<th>first</th>
<th>id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blow</td>
<td>Joe</td>
<td>3724</td>
<td>A</td>
</tr>
<tr>
<td>Blow</td>
<td>Sam</td>
<td>2475</td>
<td>B+</td>
</tr>
<tr>
<td>Crud</td>
<td>Sally</td>
<td>4237</td>
<td>A-</td>
</tr>
<tr>
<td>Smith</td>
<td>John</td>
<td>3387</td>
<td>A</td>
</tr>
<tr>
<td>Smith</td>
<td>John</td>
<td>4520</td>
<td>B-</td>
</tr>
</tbody>
</table>

The current version of the Student class is shown below. Assume the methods shown have already been implemented (i.e., don't implement them). Modify and add code to the Student class to enable you to use it in the TreeMap for the purposes described above. Hint: see code handout.

```java
public class Student {
    private String lastName;
    private String firstName;
    private int id;
    public Student(String first, String last, int studentID) { . . . }
    public String getFirst() { . . . }
    public String getLast() { . . . }
    public int getId() { . . . }
}
```
Problem 6 [8 points]
Consider the use of a stack for storing information about function activations as a program runs. As discussed in lecture this is called the run-time stack or call stack. Reminder: a stack has the operations push, pop, top, and isEmpty.

Part A. In this application calling a function corresponds to which stack operation?

Part B. In this application returning from a function corresponds to which stack operation?

Part C. Consider the following program and suppose we are currently executing at the point labeled **C**. Show two different possible contents of the run-time stack at this point (Note there are more than two correct answers). Make it clear which direction the stack is going by labeling which end is the top of each stack you draw.

```java
public class StackProb {
    public static void main(String[] args) {
        W();
        X();
        Y();
    }

    public static void W() {
        X();
        Z();
    }

    public static void X() {
        Y();
        Z();
    }

    public static void Y() {
        Z();
        // **C**
    }

    public static void Z() {
        return;
    }
}
```
Problem 7 [20 points]

Write the function removeMiddleBlob, which removes the middle section of its LinkedList parameter. For the purposes of this problem the middle section (or middle blob) means the middle element and all adjacent elements with the same value as the middle element. You may assume you start with an odd length list. Examples:

<table>
<thead>
<tr>
<th>list</th>
<th>list after call to removeMiddleBlob(list):</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4 1 5)</td>
<td>(4 5)</td>
</tr>
<tr>
<td>(7 7 4 4 4 4 5)</td>
<td>(7 7 5)</td>
</tr>
<tr>
<td>(3 4 3 3 7 3 5)</td>
<td>(3 4 7 3 5)</td>
</tr>
<tr>
<td>(3 3 4)</td>
<td>(4)</td>
</tr>
<tr>
<td>(4 3 3)</td>
<td>(4)</td>
</tr>
<tr>
<td>(3)</td>
<td>()</td>
</tr>
<tr>
<td>(3 3 3 3)</td>
<td>()</td>
</tr>
</tbody>
</table>

Hint: position an iterator at the middle of the list using the one-parameter version of the listIterator method (details near the bottom of the code handout).

```java
// removes middle section of the list as defined above
// PRE: the list is not empty and list.size() is an odd number
public static void removeMiddleBlob(LinkedList<Integer> list) {
```