This test has 14 pages (with a help page at the end), be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages are imported in any code you write.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>75 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1: (Book Shelf: 15 points)

You have a collections of books to put on your bookshelf. You have decided to order them largest to smallest by number of pages and to break ties alphabetically by title. The method sortBooks will help you, but you need to complete the implementation of the Book class for sortBooks to work. The method sortBooks has two parameters, bookTitle and pageCount which hold book titles and page counts respectively, such that the title in bookTitle[i] has pageCount[i] pages.

Example:

Input:
bookTitle = [“Catcher in the Rye”, “Adventures of Huck Finn”, “Lord of the Flies”, “Sun Also Rises”]
pageCount = [253, 312, 198, 253]
Output:
“Adventures of Huck Finn” “Catcher in the Rye” “Sun Also Rises” “Lord of the Flies”

Code: sortBooks

```java
public void sortBooks(String[] bookTitle, int[] pageCount){
    ArrayList<Book> list = new ArrayList<Book>();
    for(int i = 0; i < bookTitle.length; i++)
        list.add(new Book(bookTitle[i], pageCount[i]));
    Collections.sort(list);
    for(Book b: list)
        System.out.println(b.myTitle);
}
```

Instance variables and constructor: 4 points

Complete the implementation of class Book by adding instance variables and completing the constructor so that the class Book will work with the code given above.

```java
public class Book implements Comparable<Book>{
    // add your instance variable(s) here

    // complete the constructor definition and don’t forget to define your parameters
    public Book( )
}
```

Question continues on the next page.
Equals and hash code: 6 points
Complete the equals and hashCode methods below.

```java
    public boolean equals(Object obj){
        if (obj == this) {
            return true;
        }
        if (obj == null || obj.getClass() != this.getClass()) {
            return false;
        }
        Book temp = (Book) obj;
    }

    public int hashCode(){
    }
```

compareTo: 5 points
Complete the compareTo method below.

```java
    public int compareTo(Book arg0) {
    }
```
PROBLEM 2:  \((\text{Big-Oh: 12 points})\)

For each of the following, give the running time in terms of the parameter \(n\) in big-Oh notation. You must justify your answer. (2 points for each running time, and 1 point for each justification)

```java
public int numberOne(int n){
    int answer = 1;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n*n; j++)
            answer++;
    return answer;
}
```

Big-Oh: \(\text{Justification:} \)

```java
public int numberTwo(int n){
    int answer = 1;
    for(int i = 0; i < n; i++)
        answer++;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            answer++;
    return answer;
}
```

Big-Oh: \(\text{Justification:} \)

```java
public int numberThree(int n){
    int answer = 1;
    for(int i = 1; i <= n; i=i*2)
        answer++;
    return answer;
}
```

Big-Oh: \(\text{Justification:} \)

```java
public int numberFour(int n){
    int answer = 1;
    for(int i = 1; i <= n; i++)
        for(int j = 0; j < i; j++)
            answer++;
    return answer;
}
```

Big-Oh: \(\text{Justification:} \)
PROBLEM 3: ( Lists: 12 points)

Part A: 4 points
What is the difference between an ArrayList and a LinkedList?

What do an ArrayList and a LinkedList have in common?

Part B: 4 points
The following code removes all elements from the list passed as a parameter:

```java
public void removeAll(List<String> list){
    int theSize = list.size();
    for(int i = 0; i < theSize; i++)
        list.remove(0);
}
```

What is the running time of the call `removeAll(list1)` of n-element list1 using the declaration

`List<String> list1 = new ArrayList<String>();`

Explain your answer.

*Question continues on the next page.*
What is the running time of the call `removeAll(list2)` of n-element `list2` using the declaration

```java
List<String> list2 = new LinkedList<String>();
```

Explain your answer.

**Part C: 4 points**

The method `remove`, shown below, has the same running time as `removeAll` but does not perform in the same way.

```java
public void remove(List<String> list){
    for(int i = 0; i < list.size(); i++)
        list.remove(0);
}
```

Explain why `remove` and `removeAll` have the same running time.

If `list1` and `list2` each contain n-elements, how many elements will `list1` and `list2` have after calling `remove(list1)` and `removeAll(list2)`?
PROBLEM 4:  \((Hashing: 8 \text{ points})\)

Part A: 4 points
The following numbers are inserted into a 10 element hash table (indexed 0 - 9) in the order shown
\([2491, 5749, 3584, 1257, 5981, 3475, 5299, 5429, 5241]\)
Using the hash function \(h(x) = x \mod 10\) draw the resulting hash table using separate chaining (i.e. linked-lists).

Part B: 4 points
Adding elements to a hash table is often very fast. However, not always. Explain how insertion into a hash table can become slow. Your discussion should talk about hashCode.
Your friends decided to make a game of who can eat more fruits and vegetables to encourage healthy eating since everyone says to eat more fruits and vegetables. They determined that each player’s score for the week would be:

\[(\text{number of times most common food is eaten}) \times (\text{total number of unique foods eaten})\]

That is, if you ate "banana orange apple lettuce avocado banana apple banana" you ate bananas 3 times and you ate 5 different foods (banana orange apple lettuce avocado). Therefore, you would get \(3 \times 5 = 15\) points.

You decided to write a program to help your friends determine the winner.

**Part A: 4 points**

Complete the helper method `buildMap` which takes as a parameter a space-separated list of food someone ate, and returns a HashMap with key–food item (as a String), and value–number of times the food item was eaten.

```java
public HashMap<String, Integer> buildMap(String str){
```
Part B: 4 points
Complete the helper method `calculateScore` which takes as a parameter the HashMap built in `buildMap` and returns the player’s score as defined on the previous page.

```java
public int calculateScore(HashMap<String, Integer> map){
```

Part C: 4 points
Complete the method `whoWon` that takes parameters `food`–a String array of a space-separated lists of the food each person ate for the week, and `name`–a String array of the names of each friend who is playing, such that `name[i]` ate `food[i]`. The method `whoWon` should return the name of the person with the most points. Do not worry about breaking ties.

```java
public String whoWon(String[] food, String[] name){
```
PROBLEM 6:  (Linked lists: 16 points)

In class we used a linked-list to implement a stack. Linked lists are commonly used to implement a queue, which is how it is done in Java. For this question you will implement an integer queue using linked lists. Your queue will use the Node class below:

```java
public class Node{
    Node myNext;
    int myData;

    public Node(int data, Node next){
        myNext = next;
        myData = data;
    }
}
```

Part A: 4 points

The queue below stores the values (3,5,4) where 3 was added first to the queue, then 5, and then 4. The node myFront points to the front of the queue (from which elements are dequeued) and myBack points to the back of the queue (to which elements are added).

Draw the resulting queue after an enqueue of the value 2. That is, you will add a new node to your queue that holds the value, 2. Note: myBack.myNext == null.

After an enqueue of 2, draw the resulting queue after a dequeue is called.
Part B: 4 points
Below is a partially completed integer queue with both front and back pointers, myFront and myBack. The number of elements in the queue is held in mySize. This code uses the Node class on the previous page.

```java
public class intQueue {
    private Node myFront;
    private Node myBack;
    private int mySize;

    Complete dequeue. You can assume dequeue isn’t called unless there is at least one element in the queue.

    //remove a node from the beginning of the linked list
    public int dequeue(){
    }
}
```

Part C: 4 points
The code in the if handles an empty queue condition. Complete enqueue by adding code to the else, when the queue is not empty.

```java
//add a node to the end of the linked list
public void enqueue(int data){
    if(myBack == null){ //the queue is empty
        myBack = new Node(data, null);
        myFront = myBack;
        size++;
    }
    else{ //the queue is not empty
    }
}
```
Part D: 4 points
What is the running time (big-Oh) of the enqueue and dequeue methods?

enqueue:

dequeue:

The implementation of intQueue used both a front and back pointer. What would happen to the running time of the enqueue and dequeue methods if your queue only had a front pointer? Explain your reasoning.

enqueue:

dequeue:

Why?
String

- `.length()` Get the length of the `String`. $O(1)$.
- `.charAt(i)` Get the char at index $i$. $O(1)$.
- `.split(" ")` Split a string by spaces and store it in a `String[]`.
- `.substring(i, j)` Get the substring between indices $i$ and $j$. Index $i$ is inclusive, and index $j$ is exclusive. $O(1)$. For example:

```
String x = "abcdefg";
String y = x.substring(2, 4);
// y now has the value "cd"
```

ArrayList<T> // Where T is a type, like String or Integer

- `.add(i, X)` Add element $X$ to the list at index $i$. If no $i$ is provided, add an element to the end of the list. Adding to the end runs in $O(1)$.
- `.get(i)` Get the element at position $i$. Runs in $O(1)$.
- `.set(i, X)` Set the element at position $i$ to the value $X$. $O(1)$.
- `.size()` Get the number of elements. $O(1)$.

HashSet<T> // Where T is a type, like String or Integer

- `.size()` Compute the size. $O(1)$.
- `.add(X)` Add the value $X$ to the set. If it’s already in the set, do nothing. $O(1)$.
- `.contains(X)` Return a `boolean` indicating if $X$ is in the set. $O(1)$.
- `.remove(X)` Remove $X$ from the set. If $X$ was not in the set, do nothing. $O(1)$.

HashMap<K, V> // Where K and V are the key and value types, respectively.

- `.size()` Compute the size. $O(1)$.
- `.containsKey(X)` Determines if the map contains a value for the key $X$. To get that value, use `.get()`. $O(1)$.
- `.get(X)` Gets the value for the key $X$. If $X$ is not in the map, return `null`. $O(1)$.
- `.put(k, v)` Map the key $k$ to the value $v$. If there was already a value for $k$, replace it. $O(1)$.
- `.keySet()` Return a `Set` containing the keys in the map. Useful for iterating over. $O(1)$.

To iterate over a `HashSet<T>`, use

```
for (T v : nameOfSet) {
    // v is the current element of the set.
}
```

This can be combined with `HashMap`’s `.keySet()` to iterate over a `HashMap`. 

14