This test has 12 pages, 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. Assume that all libraries and packages we’ve discussed are imported in any code you write.

Nodes on this test are implemented using the following declaration which is nested inside a class Test in which all methods are written.

```java
public static class Node{
    public String info;
    public Node next;
    public Node(String s, Node link){
        info = s;
        next = link;
    }
}
```
PROBLEM 1:  \textit{(Whaddyaknow (18 points))}

Part A (1 points)
What is \(\log_2(1024)\) ?

Part B (5 points)
Evaluate both sums: What is \(1 + 2 + 4 + 8 + 16 + 32\)?
What is \(1 + 2 + 3 + 4 + \cdots + 19 + 20\)?

Part C (3 points)
What is the \textit{exact} value of the variable \texttt{result} below where the method \texttt{think} follows. Show reasoning for partial credit.

\begin{verbatim}
    int result = think(1000) - think(998);

    public static int think(int n){
        int sum = 0;
        for(int k=1; k <= n; k++) sum += k;
        return sum;
    }
\end{verbatim}

Part D (4 points) What is the value returned by the call \texttt{magic(802)} where the method \texttt{magic} is below. Justify your answer.

\begin{verbatim}
    public static int magic(int n){
        int sum = 0;
        for(int k=1; k <= n; k = k * 2){
            sum += k;
        }
        return sum;
    }
\end{verbatim}

Part E (2 points)
What is the runtime complexity of the call \texttt{magic(N)}? Justify your answer.

Part F (3 points)
What is the runtime complexity of the call \texttt{magic(N*N*N*N)}? Justify your answer.
PROBLEM 2: (tops, spot, ... (18 points))

The method last2first below correctly moves the last node to the front of a linked-list and returns a pointer to the new first node. Thus the call list = last2first(list) would change

("ape", "bat", "cat", "dog")

to the list

("dog", "ape", "bat", "cat")

public static Node last2first(Node list){
    if (list == null || list.next == null) return list;

    // contains at least two nodes
    Node temp = list;
    while (temp.next.next != null) {
        temp = temp.next;
    }
    temp.next.next = list;
    list = temp.next;
    temp.next = null;
    return list;
}

Part A (3 points)
What is the big-Oh complexity of last2first when executed on an N-node list? Justify your answer briefly.

Part B (3 points)
Describe the contents of the linked-list referenced by list after the code fragment below executes when list is an n-node list.

for(int k=0; k < n; k++){
    list = last2first(list);
}
Part C: Star Trap (4 points)
The method stuff below returns a one-node list unaltered. If list represents ("ape", "cat"), what is the result of executing

\[
\text{list} = \text{stuff(list)};
\]

and what is the result of executing the statement when list is ("ape", "cat", "moose")?

\[
\text{public static Node stuff(Node list){
    if (list == null || list.next == null) return list;
    // contains at least two nodes
    list = last2first(list);
    list.next = stuff(list.next);
    return list;
}}
\]

(you should have two answers to the above question).

Part D: Again Please (8 points)
Describe the runtime complexity using big-Oh of executing stuff with an N-node list and describe the list returned (its values and their order compared to the order before the method executes). Justify your answers.
PROBLEM 3: (ILST (16 points))

Part A (8 points)
In this problem assume that the Node class has a field prev and that lists are doubly-linked. When a list is sorted it is possible to determine the number of unique/different elements in $O(n)$ time for an $n$-node sorted, doubly-linked list. For example there are 4 unique values in the list

(\texttt{"one","one","one","roo","star","star","tree","tree","tree","tree"})

Write method unique that returns the number of unique values in a sorted, doubly-linked list. You may not use any sets, maps, arraylists, etc. For full credit your routine should run in $O(n)$ time for an $n$-element list.

```java
/**
 * @param list is sorted, doubly linked
 * @return the number of different/unique values in list
 */
public static int unique(Node list){
```
Part B (8 points)

Write method `alternatingList` that returns a linked list constructed from the values in an array. The linked list contains the same values in the array but the values with even indexes appear before the values with odd indexes (the order of the values with even indexes is the same as the order in the array, and the same for odd indexes). For example, for the array \([a,b,c,d,e,f,g]\) The list below would be returned (not doubly linked).

```
A -> C -> E -> G -> B -> D -> F
```

```java
/**
 * Returns alternating list.
 * @param list contains at least one element
 */
public static Node alternatingList(String[] list){
    Node first = new Node(list[0],null);
    return first;
}
```

return first;
}

```
PROBLEM 4:  *(Stored and strode (18 points))*

The code below is a solution from the *SortedFreq* APT we discussed in class. The APT text is the last page of this exam. You’ll be asked some questions about the code. The code correctly solves the APT, i.e., it’s all green.

```java
import java.util.*;

class SortByFreqs {
    public String[] sort(String[] data){
        final List<String> list = Arrays.asList(data);
        HashSet<String> unique = new HashSet<String>(list);

        String[] words = unique.toArray(new String[0]);
        Arrays.sort(words, new Comparator<String>(){
            public int compare(String o1, String o2) {
                int diff = Collections.frequency(list,o2) -
                           Collections.frequency(list,o1);
                if (diff == 0) return o1.compareTo(o2);
                else return diff;
            }
        });
        return words;
    }
}
```

**Part A (4 points)**

If the assignment to `words` above is replaced by `String[] words = data` the code compiles but no longer gets all green, i.e., in some cases the code is not correct. Describe an array `data` of five elements for which the new/modified version will still return the correct result.

**Part B (4 points)**

In the calls to `Collection.frequency` explain why the value of the result from using `o1` is subtracted from the result using `o2` and not *vice versa.*
Part C (6 points)
The big-Oh complexity of Arrays.sort is $O(n \log n)$ for an $n$-element array. What is the worst-case complexity of the code above and when is this worst case achieved?

Part D (4 points)
Would the worst-case complexity of the code above change if the body of the compare method is replaced by the following (it’s still correct):

```java
public int compare(String o1, String o2) {
    int diff = Collections.frequency(Arrays.asList(data), o2) -
               Collections.frequency(Arrays.asList(data), o1);
    if (diff == 0) return o1.compareTo(o2);
    else return diff;
}
```

Justify your answer.
The code below is from a submission to the Jotto program. The method \texttt{processHumanCount} is called when the computer is trying to guess the human player’s secret word. The human player enters the number of letters in common between the secret word and the computer’s guess. This number is passed to \texttt{processHumanCount} and play of the game proceeds.

\begin{verbatim}
public void processHumanCount(int n){
  if (n == 6) {
    myView.showModalInfo("Great! I got your word");
    stopGame();
    return;
  }
  myGuesses.remove(myLastGuess); // A
  Iterator<String> it = myGuesses.iterator();
  while (it.hasNext()) {
    String s = it.next();
    int common = JottoUtils.commonCount(myLastGuess, s);
    if (common != n) {
      it.remove();
    }
  }
  if (myGuesses.size() == 0) { // B
    showModalMessage("I give up, either I don’t know the word or\n    "you entered conflicting common counts.");
    stopGame();
  } else {
    myLastGuess = myGuesses.get(0); // C
    myView.processModelResponse(myLastGuess);
  }
}
\end{verbatim}

The method \texttt{processHumanCount} is in the class \texttt{JottoModel} in which the instance fields below are declared:

private String myLastGuess;
private ArrayList<String> myGuesses;

When a new game is started, the code below starts the game (in method \texttt{JottoModel.newGame}) when the computer guesses.

myGuesses.clear();
myGuesses.addAll(myWordList);
Collections.shuffle(myGuesses);
myLastGuess = myGuesses.get(0);
myView.processModelResponse(myLastGuess);

(questions on next page)
Part A (3 points)
If the only change to the code is to remove the call `Collections.shuffle(myGuesses)` when a new game is started what will be the effect from the human player’s perspective? Is the `shuffle` call a good idea (why)?

Part B (3 points)
Why is it necessary to use `myGuesses` rather than the inherited ArrayList `myWordList` when writing the code shown above in method `processHumanCount`?

Part C (3 points)
If the code on the line labeled 'A': `myGuesses.remove(myLastGuess)` is removed, what will the effect be from the human player’s perspective?

Part D (3 points)
Suppose code that does intelligent removal of words from consideration beyond simply removing those words with a different number of letters in common is added with a call: `doIntelligentRemoval(myGuesses)` in the method `processHumanCount`. Would this call be more appropriate before the line labeled 'B' or 'C'? Justify your answer.
APT SortByFreqs

Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you are given an array of strings and must determine how frequently the strings occur. Return an array of strings that is sorted (ordered) by frequency. The first element of the returned array is the most frequently occurring string, the last element is the least frequently occurring. Ties are broken by listing strings in lexicographic/alphabetical order. The returned array contains one occurrence of each unique string from the array parameter.

Consider these strings (quotes for clarity, they're not part of the strings).

{"apple", "pear", "cherry", "apple", "pear", "apple", "banana"}

The array returned is:

{ "apple", "pear", "banana", "cherry" }

since the most frequently occurring string is "apple" which occurs 3 times; the string "pear" occurs twice and the other strings each occur once so they are returned in alphabetical order.

Definition

- Class: SortByFreqs
- Method: sort
- Parameters: String[]
- Returns: String[]
- Method signature:

  String[] sort(String[] data)

  (be sure your method is public)

Class

```java
public class SortByFreqs
{
    public String[] sort(String[] data)
    {
        // fill in code here
    }
}
```

Notes

None

Constraints
data will contain at most 500 elements

- each element of data will contain at most 50 characters, all characters are lowercase.

**Examples**

1. 
   data = {"apple", "pear", "cherry", "apple", "pear", "apple", "banana"}
   
   Returns: {"apple", "pear", "banana", "cherry"}
   
   This is the example given above.

2. 
   data = {"a","b","c",d"}
   
   Returns: {"a","b","c",d"}

3. 
   data = {"d","c","b","a"}
   
   Returns: {"a","b","c",d"}
   
   Same as previous, each occurs once ties are broken alphabetically.

4. 
   data = {"a","a","a"}
   
   Returns {"a"}