PROBLEM 1:  (Analyze (12 points))

Give the worst case big-Oh time for the loops in terms of n and m. Assume stuff is an ArrayList<String>

A. (3 pts) What is the worst case big-Oh time?

```java
for (int k=0; k<n; k++) {
    for (int j=0; j<= k; j++) {
        stuff.add("word" + k + j);
    }
}
```

B. (3 pts) What is the worst case big-Oh time?

```java
for (int k=0; k<n; k++) {
    for (int j=0; j< m; j++) {
        stuff.add("word" + k + j);
    }
}
```

C. (3 pts) What is the worst case big-Oh time?

```java
for (int k=1; k<n; k=k*2) {
    for (int j=0; j< m; j++) {
        stuff.add("word" + k + j);
    }
}
```

D. (3 pts) What is the worst case big-Oh time?

```java
for (int k=0; k<n; k=k+2) {
    for (int j=0; j< m; j++) {
        for (int h=0; h<n; h=h+3) {
            stuff.add("word" + k + j + h);
        }
    }
}
```

PROBLEM 2:  (We need a Master Key (10 points))
Complete the method \texttt{masterKey} that has one parameter named \texttt{map}. This method adds a new key-value pair to the map. The new key is "Master" and its value is a String made up of unique occurrences of each value from the map, in sorted order separated by blanks.

For example, if all the values in the map were "B", "A", "CD", "BC", "A" and "CD", then the key ”Master” would be added to the map with the value ”A B BC CD”. (Note the new value has only one occurrence of each value from the map and those values are in sorted order).

Complete the method below.

\begin{verbatim}
    public void masterKey(Map<String, String> map) {

PROBLEM 3: \textit{(Changing the way (15 points))}

See the ArrayListHash.java file attached that we studied in class about using an ArrayList for our buckets in hashing. In that code a new item was added to the bucket at the end of the ArrayList.

For all parts of this problem, suppose we decide to change the bucket. It is still an ArrayList, but the items in the bucket must always be in sorted order by key.

A. (6 pts) Write the new \texttt{put} method that keeps the elements in the ArrayList in sorted order. Strive for efficiency. It has been started for you below.

\begin{verbatim}
    public void put(String key, int value) {
        Combo c = getCombo(key);
        if (c != null) {
            c.value = value;
        } else {
            int bucketIndex = getHash(key);
            ArrayList<Combo> list = myTable.get(bucketIndex);
        }
    }
\end{verbatim}

B. (3 pts) Do not write the code, but explain in words how to implement the new \texttt{get} method efficiently if it should change.

C. (2 pts) Suppose the size of the empty hash table is \( M = 1000000 \) and \( N = 1,000 \) items are entered into the table. Now, what is the average big-Oh time to get one value from the table?

D. (2 pts) Suppose the size of the empty hash table is 10 and we want to insert \( N = 10000000 \) items into the empty table. What is the worst case time big-Oh time to insert the \( N \) items into the table?

E. (2 pts) Suppose the size of the empty hash table is 10 and it already contains \( N \) items in the table, where \( N \) is very large. What is the worst case big-Oh time to make \( M \) queries about whether an item is in the table?
The ABC organization has many chapters in every state of the United States. In order to keep track of all the member information an ABCChapter class has been created. An ABCChapter has three kinds of information, an array of all its member names, the state and the region of the state. We would like to be able to compare ABCChapters and thus to implement Comparable. Some of its methods are shown below and on the next page.

Assume that each member is a string with "Lastname Firstname" and that the members in each chapter are stored in myMembers in alphabetical order. For example one ABC Chapter with four members might be:

State = "NC"
Region = "Eastern"
MemberList = "Astrachan Owen", "Duvall Robert" "Forbes Jeff" "Rodger Susan"

Below is the class and also on the next page. Questions start on the next page.

```java
public class ABCChapter implements Comparable<ABCChapter>{
    private String[] myMembers;
    private String myState;
    private String myRegion;

    public ABCChapter(String[] list, String state, String region) {
        myMembers = new String[list.length];
        System.arraycopy(list, 0, myMembers, 0, list.length);
        myState = state;
        myRegion = region;
    }

    /**
     * Return value that meets criteria of compareTo conventions.
     * @param chap is the ABCChapter to which this is compared
     * @return appropriate value less than zero, zero, or greater than zero
     */
    public int compareTo(ABCChapter chap) {
        // TODO implement this method
    }

    /**
     * Return true if this ABCChapter is the same as the parameter
     * @param o is the ABCChapter to which this one is compared
     * @return true if o is equal to this ABCChapter
     */
```
public boolean equals(Object o) {
    ABCChapter chap = (ABCChapter) o;
    // TODO return correct value
}

/**
 * Return a good value for this ABCChapter to be used in hashing.
 */
public int hashCode() {
    // TODO return a hash value
}

PART A. (6 pts) Implement the "equals" method for the ABCChapter class.
Two ABCChapters are equal if they have the same state, the same region, and all the
members are the same. Assume the members are stored in alphabetical order in myMembers.

    public boolean equals(Object o) {
        ABCChapter chap = (ABCChapter) o;

PART B. (6 pts) Implement the "compareTo" method for the ABCChapter class.
An ABCChapter is less than another ABCChapter if they have fewer members. If both
chapters have the same number of members, then a chapter is less than another chapter if
its state comes before it in alphabetical order. If the states are the same then a chapter is
less than another chapter if its region comes before the other region in alphabetical order. If
both the regions and the state are the same, then one chapter is less than another chapter
if the list of member names comes before the list of the other chapter member names in
alphabetical order.

    public int compareTo(ABCChapter chap) {

PART C. (4 pts) Implement the "hashCode" method for the ABCChapter class and explain
why you think it is a good hashCode function.

    public int hashCode() {

PROBLEM 5 : (It’s the Blob (8 points))

See the handout BlobModel.java that we went over in class. You may call methods in this
class. Write the new method blobCountRegion as part of this class that is given a point in
the 2D array and counts the size of the region that contains all the points connected to this point (in the four directions: up, down, left, right). A point \( p \) is in a region if its value is in the range \( \min \leq p \text{'s value} \leq \max \). Assume the 2D array is called \( \text{myGrid} \), all the entrees values are numbers greater than or equal to 0 and you can modify \( \text{myGrid} \) if you want.

For example, consider the 2D array below on the left that has 6 rows and 5 columns. The call \( \text{blobCountRegion(1,3,5,9)} \) returns 9. Starting in position row 1 and col 3, which has value 8, all valid numbers connected to 8 in the range from 5 to 9 inclusive are counted. The picture on the right below has entered in -1 in all those 9 entries so you can see them. Note there are some numbers in myGrid in the correct range, however they are not connected to the region.

```
1 2 3 2 5
6 6 7 8 4
1 7 3 5 2
2 3 6 6 4
7 1 7 2 3
5 1 4 4 1
```

```
1 2 3 2 5
-1-1-1-1 4
1-1 3-1 2
2 3-1-1 4
7 1-1 2 3
5 1 4 4 1
```

```java
protected int[][] myGrid; // 2D array has values in it, you can modify it
// other stuff not shown

protected int blobCountRegion(int row, int col, int min, int max) {

PROBLEM 6:  (How are we connected? (9 points))

Consider the \text{LinkStuff.java} we looked at in class and also provided as a handout. Below is an attempt to create \text{tripleUp} that takes as input a Node to a linked list and is suppose to return the same list with each node duplicated and now appearing three times.

For example if the linked list is the first list below, then \text{tripleUp(list)} should result in the list shown on the second line.

```
list ->"a" -> "b"
tripleUp(list) -> "a" -> "a" -> "a" -> "b" -> "b" -> "b"
```

Here is the code for a Node and attempted code for tripleUp.

```java
public class LinkStuff {

    public class Node {
        String info;
        Node next;
        public Node(String s, Node link)
        {
```
The method `tripleUp` does not work correctly. Answer the following questions about this method. Note the same code is also shown in part D.

A. (1 pt) Explain what happens when you call `tripleUp(null)`?

B. (2 pts) Suppose `p` points to a one Node list containing the value ”a”. Give the resulting list after calling `tripleUp` on this list. [REWORDED to: Give the result of calling `tripleUp(p)`]

C. (2 pts) Suppose `p` points to a two Node list containing the Nodes with values ”a” followed by ”b”. Give the resulting list after calling `tripleUp` on this list. [REWORDED to: Give the result of calling `tripleUp(p)`]

D. (4 pts) The code for `tripleUp` is shown below. Make changes to it so it will work correctly.

```java
public Node tripleUp(Node list)
{
    if (list == null) // for empty list
        return null;
    Node copy = new Node(list.info, list);
    Node copy2 = new Node(list.info, copy.next);
    list.next = tripleUp(list.next);
    return copy2;
}
```
All the names from the Friday Afternoon clubs at Dook University are stored in an array. Each element in the array is a list of names separated by a colon ":" . Each name is a first name and last name of exactly two words. Write the method LastNamesOfPopularName that returns a String with the most popular first name, followed by a colon ":" followed by all the last names for that name, with the last names separated by a single blank. Assume there is only one first name that is the most popular and all names are unique.

For example, if the array contained the three lines shown below, then the output shown below shows the most popular name is Susan and it appears with the four corresponding last names.

"Susan Smith:Jackie Long:Mary White:Susan Brandt"
"Jackie Johnson:Susan Rodger:Mary Rodger"
"Eric Long:Susan Crackers:Mary Velios:Jack Frost:Eric Lund"

Output: "Susan:Smith Brandt Rodger Crackers"

public String LastNamesOfPopularName(String[] namelists) {

CLASS methods

public class String {
    public int length ()  // length of string
    // Returns a substring of this string that begins at the specified beginIndex and
    // extends to the character at index  endIndex - 1.
    public String substring (int beginIndex, int endIndex)
    // Returns a substring of this string that begins at the specified beginIndex
    public String substring (int beginIndex)
    // Returns position of the first occurrence of str, -1 if not found
    public int indexOf (String str)
    // Returns the position of the first occurrence of str from index start
    public int indexOf (String str, int start)
    // returns character at position index
    public char charAt(int index)
    // returns true if str has the exact same characters in the same order
    public boolean equals(String str)
    // splits into Array based on delim
    String [] split(String delim)
    // returns neg value if this string comes before anotherString,
    // 0 if equal, pos value if this string comes after
    int compareTo (String anotherString)
}

public class Collections {
    // Sorts the specified list according to the order induced by the comparator
    public static void sort(List list, Comparator c)
public class Random {
    // Create a new random number generator
    public Random()
    // Returns a pseudorandom, uniformly distributed value in [0,n)
    public int nextInt(int n)
}

public class ArrayList {
    // Constructs an empty list
    public ArrayList ()
    // Returns the number of elements
    public int size ()
    // Returns element at position index
    public Object get (int index)
    // Replaces the item at position index with element.
    public Object set (int index, Object element)
    // Appends specified element
    public boolean add (Object o)
}

public class Scanner {
    // Create Scanner that reads data from a file.
    public Scanner (File file)
    // Create Scanner that reads data from a string.
    public Scanner (String str)
    // Change delimiters used to separate items
    public void useDelimiter (String characters)
    // Check if more items are available
    public boolean hasNext ()
    // Get next delimited item as a string
    public String next ()
    // Get next line as a string
    public String nextLine ()
    // Get next delimited item as an integer value
    public int nextInt ()
    // Get next delimited item as a Double value
    public double nextDouble ()
}

public class TreeSet {
    // creates an empty TreeSet
    public TreeSet()
    // adds object e to the TreeSet
    boolean add( Object e)
    // adds all elements from a collection to this set
    boolean addAll( Collection c)
    // Retains only the elements in this set that are contained
    // in the specified collection - returns true if set changed
    boolean retainAll( Collection c)
    // Removes from this set all of its elements that are contained in the specified collection
}
boolean removeAll(Collection c)
// removes all objects from the TreeSet
void clear()
// returns true if e is in the set, otherwise returns false
boolean contains(Object e)
// returns true if set is empty, otherwise returns false
boolean isEmpty()
// returns an Iterator for the set
Iterator<Object> iterator()
// removes the object e from the set
boolean remove(Object e)
// returns the number of elements in the set
int size()  
}

public class Integer implements Comparable
{

    // The smallest and largest values of type int
    public static final int MIN_VALUE
    public static final int MAX_VALUE

    // Returns the integer represented by the argument as a decimal integer.
    public static int parseInt(String s)
    // Returns a new String object representing the specified integer.
    public static String toString(int i)

    // Returns value of Integer object as an int
    public int intValue()

    // Returns true iff obj is an Integer object that contains same int value as this object.
    public boolean equals(Object obj)

    // Returns 0 if other is equal to this Integer; a value less than 0 if this Integer is less
    // than other; and a value greater than 0 if this Integer is greater than other
    public int compareTo(Integer other)
}

public class Object
{

    // Returns true iff o is the same as this object
    boolean equals(Object o)

    // Returns string representation of this object
    String toString()
}

public interface Iterator
{
    // Returns true if iteration has more elements.
    boolean hasNext()

    // Returns next element in this iteration.
    Object next()

    // Removes current element from the collection.
    void remove()
}

public class File
{
// Open a new file from the given pathname
public File(String pathname);
}

public class Map<KEY, VALUE>
{
    // Returns the number of keys in this map.
    public int size()
    // Associates given value of type VALUE with given key of type KEY in map.
    public VALUE put(KEY key, VALUE value)
    // Returns value of type VALUE associated with given key of type KEY.
    public VALUE get(KEY key)
    // Removes mapping for given key of type KEY from map, returning old value of type VALUE
    public VALUE remove(KEY key)
    // Returns set of keys in map
    public Set<KEY> keySet()
    // Returns collection of values in map
    public Collection<VALUE> values()
}

To find the size of a built-in array, use length.