PROBLEM 1:  (Analyze (8 points))

Give the worst case big-Oh time for the loops in terms of n and m.

A. (2 pts) What is the worst case big-Oh time?

```java
for (int k=0; k<n; k++) {
    for (int j=0; j<m; j=j+2) {
        count++;
    }
}
```

B. (2 pts) What is the worst case big-Oh time?

```java
for (int k=1; k<=n; k++) {
    for (int j=1; j<n; j=j*2) {
        count++;
    }
}
```

C. (2 pts) What is the worst case big-Oh time?

\[ O(100n^2 + 3n + 10m + 7) = \]

D. (2 pts) What is the worst case big-Oh time?

\[ O(\frac{1}{2}m^3 + 100m + 22) = \]

PROBLEM 2:  (What’s the output? (6 points))

```java
ArrayList<String> names = new ArrayList<String>();
TreeSet<String> namesSet = new TreeSet<String>();
names.add("Mary");
names.add("Jack");
names.add("Mary");
names.add("Jack");
namesSet.addAll(names);
System.out.println(names);
System.out.println(namesSet);

Map<Integer,String> mappings = new TreeMap<Integer,String>();
mappings.put(7, "D");
mappings.put(3, "C");
```
mappings.put(6, "B");
mappings.put(6, "A");
for (Integer num: mappings.keySet())
{
    System.out.println(num + " " + mappings.get(num));
}

List the output below here:

**PROBLEM 3 : (Who is in charge? (8 points))**

Consider the Entertainer class and the Clown class that extends it.

```java
class Entertainer {
    private String myName;
    protected String myType;
    private int myTheatreSize;

    public Entertainer(String name, String type) {
        myName = name;
        myType = type;
        myTheatreSize = 1000;
    }

    public String getName() {
        return myName;
    }

    public String getType() {
        return myType;
    }

    public int getTheatreSize() {
        return myTheatreSize;
    }

    public void setTheatreSize(int size) {
        myTheatreSize = size;
    }

    public void greeting() {
        System.out.println("Hi, I'm " + getName());
    }

    public void venue() {

```
System.out.println("Theatre size: " + myTheatreSize + " seats");

public class Clown extends Entertainer
{
    private int myShoeSize;

    public Clown(String name, String type, int shoeSize) {
        super(name, type);
        myShoeSize = shoeSize;
    }

    public String getName() {
        return "Crazy " + super.getName();
    }

    public void greeting() {
        System.out.println("Hi, I’m CRAZY " + getName());
    }

    public void shoeInfo() {
        System.out.println("I wear shoe size " + myShoeSize);
    }
}

A. (4 pts) What is the output of the following code segment?

Clown Joe = new Clown("joe", "clown", 18);
Entertainer Maria = new Entertainer("maria", "singer");

Maria.setTheatreSize(20000);
Joe.setTheatreSize(5000);

ArrayList<Entertainer> showList = new ArrayList<Entertainer>();
showList.add(Joe);
showList.add(Maria);

for (Entertainer e: showList) {
    e.greeting();
    e.venue();
}
Joe.shoeInfo();

3
B. (2 pts) At the end of the code segment above, suppose we added the following line. Explain why it would generate an error?

Maria.shoeInfo();

C. (2 pts) Suppose we add the new method `personality` to the Clown class. Explain why there is an error in the println statement.

```java
public void personality() {
    System.out.println(myName + " is outgoing ");
}
```

PROBLEM 4 :  *(How many fit in the pool? (10 points))*

Complete the method `uniqueNames` that has one Scanner parameter named `in`. This method reads names from a file and returns a list of the unique last names in alphabetical order. The data is a list of names, one name per line. The last name is the last word on the line. There is exactly one blank between words on a line. Here is a sample data file:

John Kevin Smith
Greg Smith
Fred El Mira Gumption
Chantal LaFruit
Sarah Ann Gumption

For this data file bound to the Scanner `input`, the call `uniqueNames(input)` would return the ArrayList with the elements (in this order): Gumption LaFruit Smith.

```java
public ArrayList<String> uniqueNames (Scanner in) {
```

PROBLEM 5 :  *(The biggest course (12 points))*

Complete the method `emailsLargestCourse` that has one ArrayList parameter named `all`. Each string in `all` has three pieces of information separated by colons: the course name, the name of a student and the student’s email. This method returns an ArrayList of the emails of all students registering for the course with the LARGEST enrollment. Assume there is just one course with the largest enrollment.

An example of the Strings in an ArrayList called `DukeSpring2011` is (items separated by colons):

```java
```
The call `emailsLargestCourse(DukeSpring2011)` would return the emails of CompSci 100 (the largest course enrollment), the emails are in sorted order: amj@duke.edu, fjs@duke.edu, hp@duke.edu

```java
ArrayList<String> emailsLargestCourse(ArrayList<String> all)
```

**PROBLEM 6: (There's a package delivery (18 points))**

Consider the `Package` class that has information about a package including an IDNumber for the package and an ArrayList of Cities the package has been in so far on its way to being delivered.

```java
public class Package implements Comparable<Package> {
    private String myIDNumber; // package ID number
    private ArrayList<String> myCities; // cities have been to

    public Package(String id, String city) {
        myIDNumber = id;
        myCities = new ArrayList<String>();
        myCities.add(city);
    }

    public String getIDNumber() {
        return myIDNumber;
    }

    public void changeStatus(String city) {
        myCities.add(city);
    }

    public String toString() {
        String result = myIDNumber + " " + myCities.get(myCities.size()-1);
        return result;
    }
}
```
public boolean equals(Package other) { // code not shown
}

@Override
public int compareTo(Package other) { // code not shown
}

public int hashcode() { // code not shown
}

An example of a Package might be IDNumber="1267GH", and myCities equal to "Portland, OR", "Denver, CO", "Durham, NC" showing the Package traveled through three cities.

A. (4 pts) Write the equals method for the Package class.
A Package is equal to another Package if the IDNumbers are the same and all its state variables are the same. Equals returns true if the Packages are equal, otherwise it returns false.

public boolean equals(Package other)

B. (4 pts) Write the hashcode method for the Package class making sure you write a good hashcode. Explain why your code is a good hashcode.

public int hashcode()

C. (4 pts) Write the compareTo method for the Package class.
The method compareTo returns 0 if two packages are equal, returns a negative number if the other package is greater, and returns a positive number if the other package is smaller. A Package is less than another Package if one of these is true:

1. The package’s IDNumber comes before the other package’s IDNumber in lexicographic order.

2. The IDNumbers are identical. The length of myCities is less than the length of myCities from the other Package.

3. The IDNumbers are identical. The length of both myCities is the same. The cities are compared in the order they are in the ArrayLists. For the first pair of cities that are not the same, the Package is less than the other package if that city comes before the other Package’s city.

public int compareTo(Package other)

D. (2 pts) Consider the following code that uses the Package class.
TreeSet<Package> packages = new TreeSet<Package>();
packages.add(new Package("1234AB", "Raleigh, NC"));
packages.add(new Package("6745CD", "New York, NY"));
packages.add(new Package("8457EF", "Charlotte, NC"));
packages.add(new Package("3573GH", "Lexington, KY"));

for (Package p: packages)
{
    p.changeStatus("Washington, DC");
    p.changeStatus("Baltimore, MD");
}

System.out.println(packages);

The output would be:
1234AB Baltimore, MD, 3573GH Baltimore, MD, 6745CD Baltimore, MD, 8457EF Baltimore, MD]

Explain why in the output the packages are in sorted order by IDNumber.
E. (2 pts) Explain why the output has only one city listed with each Package.
F. (2 pts) In the code above, list the method(s) from the Package class that are implicitly used and explain where these methods are used in the code.

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)
    // returns a hash code for this string
    int hashcode()
}
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.