CS/ENGRD2110: Final Exam

December 5, 2012

NAME : ______________________________

NETID: ____________

- The exam is **closed book and closed notes**. Do not begin until instructed. You have **2.5 hours**. Good luck!
- Start by writing your name and Cornell netid on top! There are **14 numbered pages**. Check now that you have all the pages.
- Web, email, etc. may not be used. Calculator with programming capabilities are not permitted—no calculators should be needed anyway. This exam is **individual work**.
- We have **scrap paper** available, so you if you are the kind of programmer who does a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam, just so that we can make sense of what you handed in!
- Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to **fit your answers easily into the space we provided**. Answers that are not concise might not receive full points.
- In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

```
POINTS:

1. Java Classes, Methods, and Types          _______ / 19

2. Lists, Stacks, and Friends                _______ / 21

3. Trees                                     _______ / 16

4. Dictionaries and Hashtables               _______ /  6

5. Graphs and Search                         _______ / 21

6. Sorting                                   _______ / 13

7. Concurrency and Threads                   _______ / 19

8. Induction and Asymptotic Complexity       _______ / 21
                                             ===========

TOTAL                                        _______ / 136
```

1 Java Classes, Methods, and Types

1. Answer the following questions with either **True** or **False**. No explanation is necessary.

9 pts.

- When a method is invoked, the number of actual arguments and the compile-time types of the arguments are used, at compile time, to determine the signature of the method that will be invoked.

- If the method that is to be invoked is an instance method, the actual method to be invoked will be determined at run time, using dynamic method lookup.

- A constructor may be made `abstract` to specify a constructor signature in the extending class.

- Private member variables can be accessed directly by any extending classes.

- The following code will compile:

```
class Privy {
   private int hidden = 1;
}
class Child extends Privy {
   Child() { hidden = 2; }
}
```

- The following code will compile:

```
class Privy2 {
   private int hidden = 1;

   class Child2 extends Privy2 {
      Child2() { hidden = 2; }
   }
}
```

- The following code will compile but will generate a runtime exception:

```
ArrayList<?> magic = new ArrayList<Double>();
magic.add(new Double(5.0));
```

- Evaluating “`1/0`” will generate a runtime exception in Java.

- The only primitive types in Java are
`int, long, float, double, boolean, void, short, char` .

2. The following program compiles and runs. What do each of the 10 calls to `f` in the `main` method print out? Write each answer to the right of the function call.

10 pts.

```
public static class MethodResolution
{
   static void println(String s) { System.out.println(s);}

   static interface Yum {
      public void f(Food x);
   }
   static abstract class Food {
      public void f(Food x)   { println("Eat something"); }
   }
   static class Plum extends Food implements Yum {
      public void f(Banana x) { println("Jerry, eat a plum"); }
      public void f(Plum x)   { println("Eat 2 plums"); }
   }
   static class Banana extends Food implements Yum {
      public void f(Banana x) { println("Eat a banana"); }
   }
   static class Sun extends Food implements Yum {
      public void f(Food x)   { println("Vitamin D!"); }
      public void f(Yum x)    { println("Yum"); }
   }

   public static void main(String[] args) {
      Plum   p  = new Plum();
      Banana b  = new Banana();
      Sun    s  = new Sun();
      Yum    yb = (Yum)b;
      Food   fp = (Food)p;
      Food   fb = (Food)b;

      p.f(b);

      p.f(fb);

      p.f(p);

      b.f(p);

      yb.f(b);

      b.f(b);

      fp.f(fp);

      fp.f(b);

      s.f(fp);

      s.f(yb);
}}
```

2 Lists, Stacks, and Friends

1. Answer the following questions with either **True** or **False**. No explanation necessary.

   3 pts.

   - Stack operations (push, pop, isEmpty) can be worst-case $O(1)$ for a linked list implementation.
   - Stack operations (push, pop, isEmpty) can be worst-case $O(1)$ for an array list implementation using a circular buffer.
   - Enqueueing and dequeueing operations are all more expensive (worst case) for a priority queue backed by a balanced heap, than for a regular queue backed by a linked list.

2. (Similar to Carrano Ch6Ex10) Suppose that instead of doubling the size of an array-based stack when it becomes full, you increase the array size by the following sequence k, $2k$, $3k$, $4k$, ... for some positive constant k.

   (a) If you have an empty stack that uses an array whose initial size is k, and you perform n pushes (assume that $n > k$), how many resize operations will be performed?

   3 pts.

   (b) What is the total cost complexity of executing n push operations? Explain your reasoning.

   5 pts.

3. **Stack-Queue-List Jamboree:** Given the usual Java interfaces for Stack, Queue, and List, consider the following program:

```
import java.util.*;
public class Jamboree
{
   public static void main(String[] args)
   {
      Stack<Integer>       S = new Stack<Integer>();      //  push(I), pop()
      Queue<Integer>       Q = new LinkedList<Integer>(); // offer(I), poll()
      LinkedList<Integer>  L = new LinkedList<Integer>();

      int[] examJoy = {5,3,4,1,2};

      for(int i=0; i<5; i++) {
         int x = examJoy[i];
         if(i%2==0)
            L.addFirst(x);
         else
            S.push(x);
      }

      while(!S.isEmpty() || !L.isEmpty()) {
         if(!S.isEmpty()) Q.offer(S.pop());                        //enqueue
         if(!L.isEmpty()) Q.offer(L.removeLast());                 //enqueue
         if(!Q.isEmpty()) System.out.println("Joy is "+Q.poll()); //dequeue
      }
   }
}
```

(a) The program compiles, and you run it. What is printed out?

5 pts.

(b) What would be printed if Q were replaced by a "`new PriorityQueue<Integer>()`" (whose `poll()` returns the smallest enqueued item)?

5 pts.

3 Trees

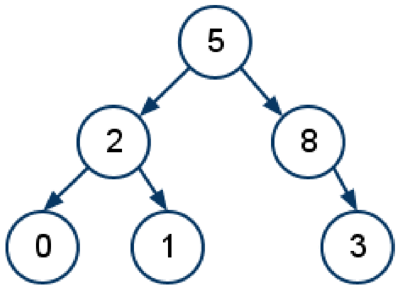
1. Answer the following questions with either **True** or **False**. No explanation necessary. 5 pts.

   - A binary search tree with n integer entries can be searched in worst case time $O(\log n)$.
   - In the tree below, nodes 5 and 2 are ancestors of 3.
   - In the tree below, node 2 is at depth 2.
   - In the tree below, nodes 0, 1 and 3 are siblings.
   - In the tree below, node 5 is at height 2.

2. Consider the following TreeNode class, and the binary tree example:

```
public class TreeNode
{
  public TreeNode left;
  public TreeNode right;
  public Integer  value;

  public TreeNode(int value) {
    this.value = new Integer(value);
  }
}
```



   - Implement a recursive `TreeNode` method that counts the number of times a value occurs in the subtree rooted at that node. Let this `TreeNode` method have the following signature: 6 pts.

     `int count(int value) {`

   - Report the pre-order traversal of the example tree, including `null` for missing children, that would be used, e.g., to write the tree to disk.

5 pts.

4 Dictionaries and Hashtables

1. Answer the following questions with either **True** or **False**. No explanation necessary.

   6 pts.

   - The worst-case complexity of checking for an object in a hash set, i.e., `contains(x)`, is $O(1)$
   - The worst-case complexity of finding a key in a hash table with n entries is $O(\lambda)$ where λ is the load factor.
   - For a hash table of size 5, the function "x % 5" maps the values $\{20, 6, 18, 14\}$ to indices without collisions.
   - For hash tables with linked-list chaining, the *load factor* λ corresponds to the fraction of buckets in the hash table that are not empty.
   - Hash tables with linear probing only work for $\lambda \leq 1$.
   - Hash tables with linked-list chaining only work for $\lambda \leq 1$.

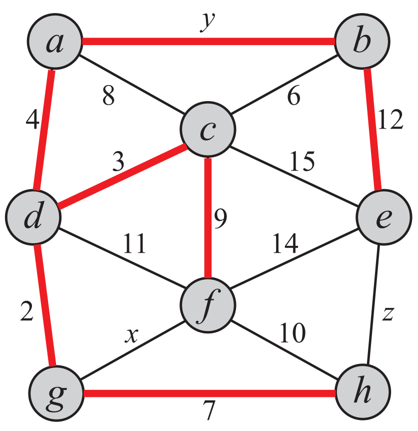
5 Graphs and Search

1. Answer the following questions with either **True** or **False**. No explanation necessary. 2 pts.

   - Minimum spanning trees can be colored with only two colors.
   - Dijkstra's algorithm is asymptotically faster than Prim's algorithm.

2. **MST Mystery:** Consider the following graph with integer edge weights, some of which are unknowns indicated by x, y, and z. The minimum spanning tree (MST) is also drawn on the graph using bold lines. Use what you know about MST algorithms to infer what the missing edge weights could be in order for the MST to be as shown. If you cannot determine the integer weights exactly, state inequality bounds on x, y and z.

   6 pts.



3. **For the Swarm!** The order in which certain Zerg structures are built in *StarCraft 2* (the "Zerg Tech Tree") can be specified as an DAG (with unweighted edges) in adjacency matrix form:

```
Hatchery         --> SpawningPool, EvolutionChamber
SpawningPool     --> Lair, SpineCrawler, HydraliskDen, BanelingNest
EvolutionChamber --> SporeCrawler
Lair             --> RoachWarren, InfestationPit, Spire
Spire            --> GreaterSpire
InfestationPit   --> Hive
Hive             --> UltraliskCavern, GreaterSpire
```

- Draw the DAG *(you can use shortened names if you wish, e.g., IP for InfestationPit)* 5 pts.

- Is this "Zerg tech tree" a tree? Why? 2 pts.

- What is the *shortest* path from `Hatchery` to `GreaterSpire`? If it is nonunique, report all shortest paths. Report each "build order" in the form "`Hatchery, Node2, ..., GreaterSpire`." 6 pts.

6 Sorting

1. Answer the following questions with either **True** or **False**. No explanation necessary. 3 pts.

   - Binary search on a sorted array is $O(\log n)$ in the worst case.
   - Merge Sort requires less memory than Quick Sort.
   - Merge Sort is faster than Quick Sort in worst-case scenarios.

2. **Counting Sort** (Carrano Ch9Ex11): A counting sort is a simple way to sort an array of n positive integers that lie between 0 and m, inclusive. You need $m + 1$ counters. Then, making only one pass through the array, you count the number of times each integer occurs in the array. From the counters, you know how many times each number occurs, and can thus determine the sorted array.

   (a) Implement the following method that performs counting sort on `x`, and returns the sorted result by over-writing `x`, i.e., `x` will be sorted after calling this method: 8 pts.
   ```
   void countingSort(int[] x, int m) {
   ```

   (b) What is the time complexity of this sorting method? 2 pts.

7 Concurrency and Threads

1. Answer the following questions with either **True** or **False**. No explanation necessary. 5 pts.

   - Every `Object` (and its subtypes) have an intrinsic lock associated with them.
   - `synchronized` statements are required to be evaluated by multiple threads at the same time.
   - A thread invoking an object's `synchronized` method may also invoke other `synchronized` methods on the same object.
   - Calling `Thread.sleep(long millis)` puts the `main` thread to sleep.
   - Threads that are not able to execute because other threads are keeping the CPU busy are said to be experiencing "deadlock."

2. **Thread-Safety First:** Suppose that you have a large number of `int[]` arrays for which you would like to compute the sum of all entries. You decide to use thread-parallel computation for speed. Your assistant comes up with the following `Thread` implementation that accumulates the sum of one `int[]` into a global variable, `sumTotal[0]`. Unfortunately he/she can not decide which of the 5 sum methods to use, but hopefully you can help.

```
static class SumTask extends Thread
{
   static  int[] sumTotal = new int[1];
   private int[] v;

   SumTask(int[] v) { this.v = v; }

   public void run() {
      /// WHICH sum?() METHOD TO USE???
   }

   void sum1() {
      for(int x : v) sumTotal[0] = sumTotal[0] + x;
   }

   void sum2() {
      synchronized(sumTotal) {
         for(int x : v) sumTotal[0] = sumTotal[0] + x;
      }
   }

   synchronized void sum3() {
      int sum = 0;
      for(int x : v) sum += x;
      sumTotal[0] = sumTotal[0] + sum;
   }

   void sum4() {
      int sum = 0;
      for(int x : v) sum += x;
      synchronized(this) {
         sumTotal[0] = sumTotal[0] + sum;
      }
   }
   void sum5() {
      int sum = 0;
      for(int x : v) sum += x;
      synchronized(sumTotal) {
         sumTotal[0] = sumTotal[0] + sum;
      }
   }
}
```

Assuming that many `SumTask` threads are launched in parallel to compute `sumTotal`:

(a) Which `sum?()` method should be used to have the best chance of *correctness* and parallel *performance*?

2 pts.

(b) Briefly explain why your choice is best.

4 pts.

3. **Wait/Notify & Synchronization:** Suppose that you decide to start your own internet service provider. You plan to have millions of users, but unfortunately you can only support 5 connections initially. Suppose that you store their Java representations in a List as follows:

```
public class ConnectionPool {
  private List<Connection> connections = createConnections(); // List not threadsafe

  private List<Connection> createConnections() {
    List<Connection> conns = new ArrayList<Connection>(5);
    for (int i = 0; i < 5; i++) {
      ... add a Connection to conns
    }
    return conns;
  }
}
```

You plan to resolve your millions of users' requests for a connection by having them call this method:

```
public Connection getConnection() throws InterruptedException {
  synchronized (connections) {
    while (connections.isEmpty()) {
      connections.wait();
    }
  }
  return connections.remove(0);
}
```

They will return unused connections using this method:

```
public void returnConnection(Connection conn) {
  synchronized (this) {
    connections.notify();
    connections.add(conn);
  }
}
```

Unfortunately both of these latter two methods have problems, and you risk losing millions of customers if you don't fix them:

(a) Something is wrong with `getConnection()`. Write a better body for this method.

4 pts.

(b) Two things are wrong with `returnConnection(...)`. Write a better body for this method.

4 pts.

8 Asymptotic Complexity and Induction

1. Answer each of the following questions by stating the computational complexity in Big Oh notation. No explanation necessary. Report the tightest bound you can, e.g., do not report $O(n^2)$ when $O(n)$ will do.

   8 pts.

   - (Carrano Ch4Q8): The following algorithm discovers whether an array contains duplicates within its first n elements. What is the worst-case cost of this algorithm?

```
boolean hasDuplicates(int[] array, int n)
{
  for(int index=0; index<=n-2; index++) {
    for(int rest=index+1; rest<=n-1; rest++) {
      if(array[index] == array[rest])
      return true;
    }
  }
  return false
}
```

   - (Carrano Ch4Ex10) What is the cost of the following computation?

```
int sum = 0;
for(int counter=n; counter>0; counter=counter-2)
  sum = sum + counter;
```

   - (Carrano Ch4Ex11) What is the cost of the following computation?

```
int sum = 0;
for(int counter=0; counter<n; counter=counter*2)
  sum = sum + counter;
```

   - What is the cost of the following computation?

```
int sum = 0;
for(int i=0; i<n; i+=100) {
  for(int j=0; j<1000*n; j++) {
    for(int k=0; k<j/1000; k++) {
      for(int l=0; l<1000000; l++) {
        sum += i+j+k-l;
      }
    }
  }
}
```

2. **Mathematical Induction:** Assume that you have a recursive algorithm that has worst-case time complexity bounded by the following recurrence relation.

$$\begin{aligned} T(1) &= 2 \\ T(n) &= T(n-1) + 3(n-1)^2 \end{aligned}$$

In the following you will use weak mathematical induction to prove that the algorithm is $O(n^3)$.

- State and prove the Base Case.

3 pts.

- State the Inductive Hypothesis.

4 pts.

- Perform the Induction Step. Use the back of this page if you need more room. *(Simplifying hint: $((n+1)-1) = n$)*

6 pts.