CS/ENGRD2110: Final Exam

11th of May, 2012

NAME : _____

NETID: _____

- The exam is **closed book and closed notes**. Do not begin until instructed. You have **150 minutes**.
- 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. This exam is **individual work**.
- We have **scrap paper** available. 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. **Note that scrap paper will be DISCARDED and NOT GRADED.**
- 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. It you do need more space, use the back page of the exam.

POINTS:

Java Classes, Methods, and Types	_____ / 17
Lists, Stacks, and Friends	_____ / 20
Dictionaries and Hashtables	_____ / 15
Graphs	_____ / 29
Graph Search	_____ / 7
Concurrency and Threads	_____ / 14
Induction and Asymptotic Complexity	_____ / 10
	==========
Total	_____ /112

1 Classes, Interfaces, and Types

1. Answer the following questions with either true or false. No explanation necessary.

   5 pts.

   - If I is an interface, you can never have an object with dynamic type I.
   - A cast affects the static type, but not the dynamic type.
   - The dynamic type of an object is determined when the object is created and can never be changed.
   - *int* is a primitive type.
   - Upcasts can never produce runtime errors.

2. Write next to each method call in "main()" the output that it prints.

   7 pts.

```
public interface I {
    public void f(I i);
}

public class A {
    public void f(A a) { System.out.println("fa(A)"); }
    public void f(B b) { System.out.println("fa(B)"); }
}

public class B extends A implements I {
    public void f(I i) { System.out.println("fb(I)"); }
    public void f(A a) { System.out.println("fb(A)"); }
    public void f(B b) { System.out.println("fb(B)"); }
}

public class TypeMeister {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        A ab=(A)b;
        I ib=(I)b;
        // Write output next to each of the following:

        a.f(a);

        a.f(b);

        a.f(ab);

        ab.f(a);

        ab.f(b);

        ib.f(b);

        ib.f(ib);
    }
}
```

3. Name two differences between Interfaces and Abstract Classes in Java.

2 pts.

4. Given the interface and class definitions from below, what are the methods that you definitely need to implement yourself in class `MyClass`?

3 pts.

```
interface L {
        public float mL(int a);
}

interface J {
        public int mJ(int a);
        public void mC(int a);
}

abstract class C implements L {
        public void mC(int a) {
           System.out.println(``hello world'');
        }
}

class MyClass extends C implements J {
...
}
```

2 Lists, Stacks, and Trees

1. Answer the following questions with either true or false. Assume there are n elements in the datastructure. No explanation necessary.

   7 pts.

   - One can implement a queue based on a linked list so that EVERY INDIVIDUAL add/poll operation is time $O(1)$.
   - One can implement a queue (of unbounded size) based on an array so that EVERY INDIVIDUAL add/poll operation is time $O(1)$.
   - The core datastructure of Breadth-First Search is a queue.
   - One can check whether a given item is contained in a linked list of length n in time $O(\log(n))$.
   - Removing an `Integer` from a singly-linked list of `Integer` can be done in worst-case time $O(n)$.
   - Removing an `Integer` from a doubly-linked list of `Integer` can be done in worst-case time $O(1)$.
   - A binary tree of height h cannot contain more than h^2 leaves.
   - Inserting an item into an unbalanced binary search tree with n elements may require time $O(n)$.

2. A binary search tree produces the following preorder traversal, where "null" indicates an empty subtree (i.e. the left/right child is the null pointer).

   9,5,3,1,null,null,4,null,null,8,6,null,null,null,20,12,10,null,11,null,null,null,30,21,null,null,31,null,null

   Draw the tree that produced this preorder traversal.

   4 pts.

3. Draw the Binary Search Tree (BST) after inserting the following sequence of elements. Assume that smaller elements are on the left.

   6,4,8,1,12,3,19,5,20

   Draw the final BST after all insertions are completed.

   4 pts.

4. Write a recursive method `static int sizeOfTree(Node root)` that counts the number of nodes in a binary tree with the given root node. Do not use external fields to store intermediate results.

   For this question, assume you have a `Node` class that has the basic methods implemented: getLeft(), getRight(), setLeft(), setRight()...

   5 pts.

```
static int sizeOfTree(node root) {
```

3 Dictionaries and Hashtables

1. Answer the following questions with either true or false. Assume there are n elements in the datastructure and that a and b are of type Object. No explanation necessary.

   5 pts.

   - If a and b have the same hash code then a.equals(b) always returns true.
   - Quadratic probing does not work when the load factor is greater than 1.
   - When inserting n elements into a hash table, the total number of copy-operations over all iterations of table doubling does not exceed $2n$.
   - If you create a subclass S of Object that overrides (and changes) the equals() method, the method hashCode() should also be overridden in order to be able to use `HashSet<T>` (or other hashtables).
   - The runtime for insert into a hash table is $O(1)$ even with a bad hash function.

2. You have two Dictionary implementations, one using a sorted array, one using an unsorted array. Assume that you know ahead of time what the maximum number of key-value pairs is that you will ever need to store at any point in time (i.e. you never increase the size of the array). For the following operations, please write "sorted", "unsorted", or "same" to indicate which implementation has the faster worst-case runtime (or if they are the same).

   4 pts.

   - Insert a new key-value pair:
   - Given a key, remove a key-value pair:
   - Look up the value for a given key:
   - Update the value for a given key:

3. Write a method that returns TRUE if an `ArrayList<Integer>` of (positive and negative) integers contains three numbers that sum to 0, and FALSE otherwise. Assume that the list contains no duplicate elements. Your method has to run in $O(n^2)$ time.

   Hint: Make use of `HashSet<T>`, which lets you execute `insert(<T>)` and `contains(<T>)` in amortized time $O(1)$. Remember that `get(i)` returns the i-th element of an `ArrayList`.

   6 pts.

```
public boolean threeElementsSumZero(ArrayList<Integer> lst) {
```

4 Graphs

1. Answer the following questions with either true or false.

   6 pts.

   - Prim's and Kruskal's algorithms will always return the same Minimum Spanning tree (MST).
   - Prim's algorithm for computing the MST only work if the weights are positive.
   - An MST for a connected graph has exactly V-1 edges, V being the number of vertices in the graph.
   - A graph where every edge weight is unique (there are no two edges with the same weight) has a unique MST.
   - The MST can be used to find the shortest path between two vertices.
   - Any bipartite graph can be colored with three colors.

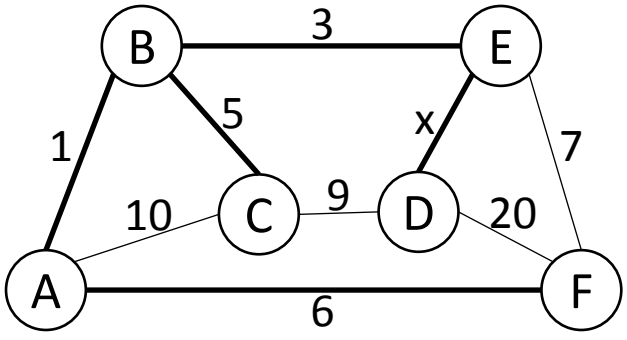
2. Is the graph consisting of the following set of edges planar? Give a graphical justification for your answer.

   4 pts.

   (A,C) (A,E) (B,A) (B,D) (B,F) (C,B) (D,A) (D,E) (F,A) (F,C) (F,E)

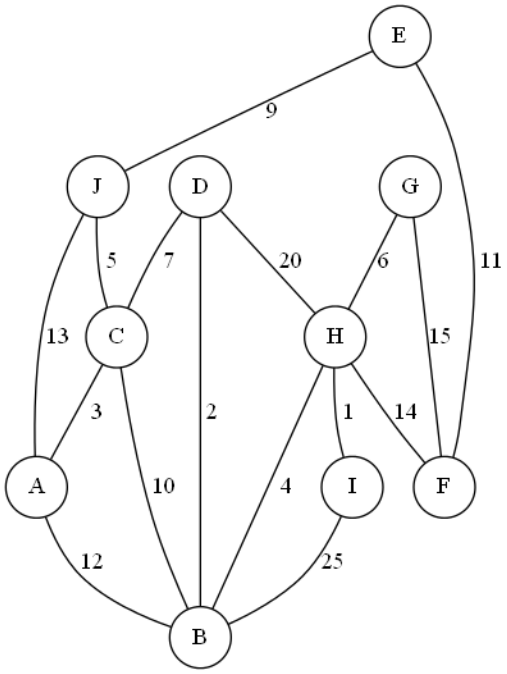
3. For the following graph the bold edges form a Minimum Spanning Tree. What can you tell about the range of values for x?

   3 pts.



4. Use Prim's algorithm starting at node A to compute the Minimum Spanning Tree (MST) of the following graph. In particular, write down the edges of the MST in the order in which Prim's algorithm adds them to the MST. Use the format $(node_1, node_2)$ to denote an edge.

5 pts.



5. For the same graph as above, write down the edges of the MST in the order in which Kruskal's algorithm adds them to the MST.

5 pts.

6. Describe in words (or pseudo-code) an algorithm for updating the MST of a graph when an edge (a, b) is deleted from the MST (and the underlying graph). It's time complexity must be better than running an MST algorithm from scratch. State and explain the time complexity of your algorithm.

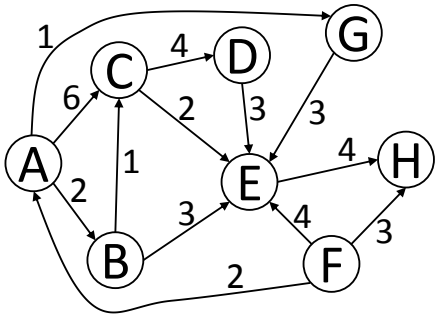
   You can assume that all edge weights are distinct, that the graph has E edges and V vertices after the deletion, that the graph is still connected after the deletion of the edge, and that your graph and your MST are represented using adjacency lists.

   6 pts.

5 Graph Search

1. For the graph below, show how the BFS Algorithm for Weighted Graphs (aka Dijkstra) finds the shortest path from A to H. For each iteration, show the content of the priority queue. The priority queue contains triples (to,from,cost). Note that multiple triples with the same `to` node can occur in the priority queue. However, do not add another triple if you have already encountered a better path to `to` (e.g. you keep a hashtable with the cost of the best path to `to` so far).

7 pts.



6 Concurrency and Threads

1. Select the answer below which best fits: Thread A and Thread B are concurrent threads that access and perform operations on an object C. When running the program multiple times, you notice that the program terminates with different results. What do you suspect is the problem:

   2 pts.

   - Deadlock
   - Mutual exclusion
   - Livelock
   - Race condition

2. Select the answer below which best fits: You create a GUI with separate threads. One thread draws the screen, one handles user input, and one handles sounds. The user clicks the close button and we start closing the program. The application halts when ActionListeners send their last ActionEvents and the GUI finishes painting, but the sound thread stops in the middle of its song without finishing. This is because:

   2 pts.

   - The sound thread didn't hold the lock when the user closed the program.
   - The sound thread was never notified.
   - The sound thread was a daemon thread.
   - The sound thread is now in a deadlock.

3. Deadlocks can cause problems in multi-threaded programs. Fortunately, there are some general strategies to ensure that deadlocks cannot occur. Examples of these are (circle all that apply) :

   2 pts.

   - Limit threads to holding only one lock at a time.
   - Create special objects that are only used for locking, but not for storing any data.
   - Limit threads to lock objects in a set order.
   - Limit threads to release locks in a set order.
   - Limit number of threads concurrently accessing a shared resource to 2.

4. Answer the following questions with either true or false. No explanation necessary.

   4 pts.

   - Threads cannot access objects that are currently used by a different thread.
   - Private fields in objects cannot be accessed from by threads other than the thread that created the object.
   - `synchronized(o)` locks an object `o` so that no other thread could possibly access it.
   - When a Java thread starts, it executes the method run().

5. The following code suffers from a potential deadlock. Show an execution ordering in which the deadlock occurs. In particular, show in which order the threads `t1` and `t2` call the methods `tj.smile(am)`, `am.smile(tj)`, `tj.waveBack(am)`, and `am.waveBack(tj)`, as well as who holds/waits for locks on which object at any time.

4 pts.

```
class Person{
   String name;
   public Person(String name){
      this.name = name;
   }
   public synchronized void smile(Person p){
      System.out.println(name + " smiles at " + p.name);
      p.waveBack(this);
   }
   public synchronized void waveBack(Person p){
      System.out.println(name + " waves back at " + p.name);
   }
}

class T1 extends Thread{
   Person p1;
   Person p2;
   public T1 (Person p1, Person p2){
      this.p1 = p1;
      this.p2 = p2;
   }
   public void run(){
      p1.smile(p2);
   }
}

public class one {
   public static void main(String[] args){
      Person tj = new Person("Thorsten Joachims");
      Person am = new Person("Andrew Myers");
      T1 t1= new T1(tj, am);
      T1 t2 = new T1(am, tj);
      t1.start();
      t2.start();
   }
}
```

Active Thread	Method Call	Locks Held (e.g. t2 locks o)	Lock Wait (e.g. t1 waits for lock on o)

7 Induction and Asymptotic Complexity

1. Give the definition of "$f(n)$ is $O(g(n))$".

   4 pts.

2. Assume you have a recursive algorithm that has worst-case time complexity bounded by the following recurrence relation. Prove by induction that the algorithm is $O(\log_2(n))$. Explicitly state the Base Case, the Inductive Hypothesis, and the Induction Step.

   6 pts.

   $T(1) = 0$
   $T(n) = T(\lfloor n/2 \rfloor) + 1$ for $n \geq 2$
   Note that the floor function $\lfloor a \rfloor$ rounds a downwards (i.e. it always holds that $\lfloor a \rfloor \leq a$).

   Base Case:

   Inductive Hypothesis:

   Induction Step: