CS/ENGRD2110: Final Exam
SOLUTION

12th of May, 2011

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 **17 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.
- 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	_______ / 19
Lists, Stacks, and Friends	_______ / 17
Trees and BSTs	_______ / 19
Dictionaries and Hashtables	_______ / 9
Graphs	_______ / 26
Graph Search	_______ / 14
Sorting	_______ / 10
Concurrency and Threads	_______ / 15
Induction and Asymptotic Complexity	_______ / 15
	===========
Total	_______ /144

1 Classes, Interfaces, and Types

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

   7 pts.

   - Both interfaces and classes define the type system in Java.
   - A type in Java can have more than one supertype.
   - A type in Java can have more than one subtypes.
   - A cast can be used to change the static type of a local variable.
   - Downcasts can produce runtime errors.
   - An abstract class cannot contain any method implementations.
   - The dynamic type of an argument to an overloaded method determines which of the methods is selected.

   SOLUTION:
   yes,yes,yes,yes,yes,no,no
   END SOLUTION

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

   9 pts.

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

class B extends A {
    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 ba=(A)b;
                // Write output next to each of the following:

                a.f(a);

                a.f(b);

                b.f(a);

                b.f(b);

                a.f(ba);

                b.f(ba);

                ba.f(a);
                ba.f(b);

                ba.f(ba);
        }
}
```

SOLUTION:
fa(A), fa(B), fb(A), fb(B), fa(A), fb(A), fb(A), fb(B), fb(A)
END SOLUTION

3. 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 I {
        public float mI(int a);
}

interface J extends I {
        public int mJ(int a);
        public Object mJJ(int a);
}

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

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

SOLUTION:
mI, mJ, mJJ
END SOLUTION

2 Lists, Stacks, and Friends

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 stack based on a linked list so that EACH INDIVIDUAL push/pop operation is time $O(1)$.
   - One can implement a stack (of unbounded size) based on an array so that each individual push/pop operation is time $O(1)$.
   - The core datastructure of Depth-First Search is a queue.
   - One can reverse the order of the elements in a linked list in time $O(n)$.
   - It is possible to append two linked lists in time $O(1)$.
   - Adding an element to a heap has worst-case time complexity $O(log(n))$.
   - Returning the maximum element in a max-heap (but not deleting it from the heap) can be done in time $O(1)$.

   SOLUTION:
   true,false,false,true,true,true,true
   END SOLUTION

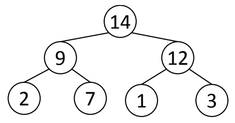
2. Construct a balanced binary max-heap (i.e. a heap that always returns the maximum element) using the following elements, pushing them onto the heap in the given order:

   7, 2, 1, 9, 12, 3, 14

   Draw the heap after each completed insertion of an element.

   6 pts.

   SOLUTION:
   This is the correct final heap:

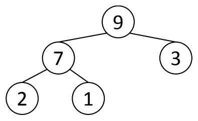


   END SOLUTION

3. Now pop (i.e. extract) the two largest elements off the heap. Draw the heap after each such extraction.

   4 pts.

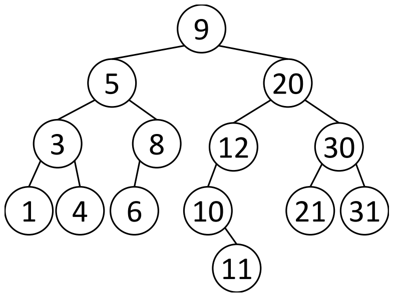
   SOLUTION:
   This is the correct final heap:



END SOLUTION

3 Trees and BSTs

1. Give the preorder, inorder, and postorder traversal of the following tree.



6 pts.

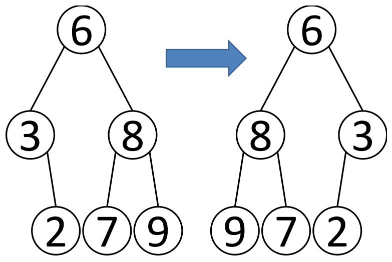
SOLUTION:
Preorder: 9,5,3,1,4,8,6,20,12,10,11,30,21,31
Inorder: 1,3,4,5,6,8,9,10,11,12,20,21,30,31
Postorder: 1,4,3,6,8,5,11,10,12,21,31,30,20,9
END SOLUTION

2. You have a binary search tree (BST) with n elements that has height $h = O(log(n))$, and you need to find the k-th largest element in the tree. Can one find the k-th largest element without scanning through all n elements (assuming $k < n$)? If yes, describe an algorithm (no code, just english). If not, provide a counterexample.

6 pts.

SOLUTION:
Yes. Do inorder traversal and stop at the k-th node. This takes time $O(h + k)$.
END SOLUTION

3. Write a recursive method `static void mirrorTree(node root)` that changes a given input tree so that it becomes the mirror image of the original tree. For example:



For this question, assume you have a node class that has the basic methods implemented: getLeft(), getRight(), setLeft(), setRight(), getValue(). All the values in a node are integers.

7 pts.

SOLUTION:
```
static void mirrorTree(node root) {
        if(root == null) {
                return;
        }
        node left = root.getLeft();
        node right = root.getRight();
        node.setLeft(right);
        node.setRight(left);
        modTree(right);
        modTree(left);
}
```
END SOLUTION

4 Dictionaries and Hashtables

1. Chaining and probing are two methods used to resolve collisions in hash tables so that the amortized access time is $O(1)$. For each of the following claims, indicate whether it is true of **chaining**, **probing**, **both**, or **neither**.

   5 pts.

   - Needs additional memory beyond the primary array for the hash table.
   - Requires doubling the table periodically.
   - May be either "linear" or "quadratic".
   - Crashes if the load factor become greater than 1.
   - Requires computing the hash function multiple times when doing an insertion.

   SOLUTION:
   Chaining,Both,Probing,Probing,Neither.
   END SOLUTION

2. In order to utilize the predefined Java classes HashMap and HashSet, what two methods inherited from class Object might need to be overridden?

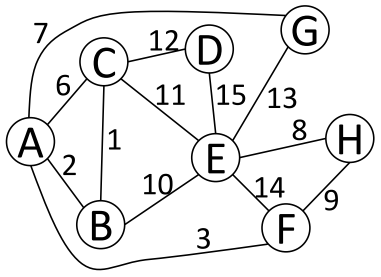
   4 pts.

   SOLUTION:
   hashCode() and equals()
   END SOLUTION

5 Graphs

1. 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.

7 pts.



SOLUTION:
The following edges are added to the MST in the given ordering: (A,B), (B,C), (A,F), (A,G), (F,H), (H,E), (C,D)
END SOLUTION

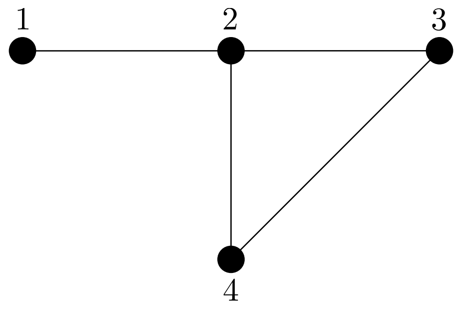
2. Given a graph $G = (V, E)$, arbitrarily partition the nodes into two disjoint sets, V_1 and V_2. Let E_1 be all the edges such that both nodes in the edge are in V_1; let E_2 be all edges such that both nodes are in V_2; let E_3 be all edges (u, v) such that $u \in V_1$ and $v \in V_2$. If we construct a Minimum Spanning Tree M_1 on (V_1, E_1) and a Minimum Spanning Tree M_2 on (V_2, E_2), then connect M_1 and M_2 on the lowest-weighted edge connecting M_1 and M_2, will it be a Minimum Spanning Tree of G? Give a proof that the algorithm correctly computes the Minimum Spanning Tree, or give a counterexample that it does not.

5 pts.

SOLUTION:
This algorithm does NOT produce the MST of G. A simple counterexample: 4 nodes a,b,c,d with $w(a, b) = 100$, $w(b, c) = 1$, $w(c, d) = 2$, $w(d, a) = 3$. Partition the nodes into $V_1 = \{a, b\}$ and $V_2 = \{c, d\}$. Then $w(M_1) = 100$, $w(M_2) = 2$, and the weight of the spanning tree over G is 103. But the MST has weight 6.
END SOLUTION

3. Write down the adjacency matrix A of the following undirected graph. Note that each undirected edge corresponds to two directed edges of weight 1.

4 pts.



SOLUTION:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

END SOLUTION

4. Let P_{ij} be the number of paths of length two in the above graph that start from vertex i and finish in vertex j. For example, $P_{23} = 1$ because there is only one path of length two that connects 2 and 3: 2—4—3. The same edge can be used many times in each path (i.e. 2—4—2 is a path). Write down the matrix P, i.e. the number of paths of length 2 for each pair of vertices.

4 pts.

SOLUTION:

$$P = A^2 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

END SOLUTION

5. Describe in words an algorithm for computing the number of paths of length l between two given vertices i and j. The graph is unweighted and you know its adjacency matrix A. State the runtime of your algorithm in Big-O notation and explain why your algorithm has the specified runtime.
NOTE: Any correct algorithm will get points independent of its efficiency, but for full points your algorithm should be logarithmic in l and polynomial in the number of vertices V.

6 pts.

SOLUTION:
A straightforward algorithm is to recursively explore all paths from i of length l. When the depth l of the recursion is reached, check whether the current node is j. If it is, increment a counter. At the end, the counter countains the number of paths. This algorithm has runtime $O(V^l)$, since each vertex can has V neighbors.

For a faster algorithm, note that the i, j-th element of $P = A^l$ contains the number of paths of length l between two given vertices i and j. Compute $P = A^l$ using repeated squaring. Then return P_{ij}.

To compute A^l by repeated squaring we need $\log(l)$ matrix multiplications each of which takes $O(|V|^3)$ using the standard algorithm (Strassen's $O(|V|^{\log(7)}$ algorithm would be even faster). So, the overall runtime is $O(|V|^3 \log(l))$.
END SOLUTION

6 Graph Search

1. Give a pseudo-code implementation of a function `bfs_path(G,s,t,max)` that uses Breadth-First Search (BFS) to return true if an arbitrary weighted graph G contains a path from s to t that has cost less or equal to max, and that otherwise returns false. Indicate which datastructures you are using. You can assume that standard datastructures are available and that $s \neq t$.

7 pts.

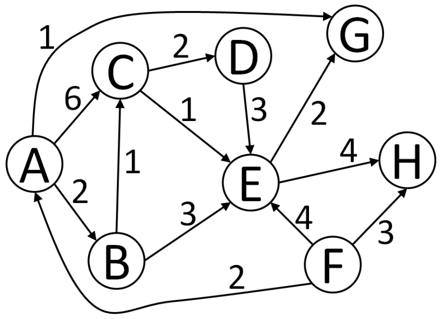
SOLUTION:

```
• Input: start node $s$, destination node $t$, max cost $max$
• Put start node $(s, 0)$ into priority queue with cost 0 and mark $s$ as visited (use e.g. Hashset).
• While priority queue not empty
    – Poll $(n, c)$ off priority queue.
    – Mark $n$ as visited.
    – IF $c > max$ THEN return FALSE
    – IF $n$ equals $t$ THEN return TRUE
    – FOR all (unmarked) successors $n'$ of $n$
        ∗ Put $(n', c + edgecost)$ into priority queue
• return FALSE
```

NOTE: An alternative implementation is to already return true when adding the target node with cost less than v to the priority queue. Another optimization would be to not add any nodes to the queue where the current path cost is greater than max.
END SOLUTION

2. In the graph below, use your algorithm from above to compute whether there is a path from node A to node E that has cost of at most 4. In particular, whenever BFS expands a new node, show the content of the main datastructure that BFS maintains. Break ties arbitrarily.

7 pts.



SOLUTION:
The main datastructure is the Queue, but for completeness I also added which nodes are already marked as visited:
(Queue: A; Visited: none)
Queue: (G,1),(B,2),(C,6); Visited: A
Queue: (B,2),(C,6); Visited: A,G
Queue: (C,3),(E,5),(C,6); Visited: A,G
Queue: (E,4),(D,5),(E,5),(C,6); Visited: A,G,B

END SOLUTION

7 Sorting

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

   5 pts.

   - HeapSort has worst-case time complexity of $O(n\log(n))$.
   - HeapSort makes no more than $O(n^2)$ pairwise comparisons.
   - MergeSort has best-case time complexity of $O(n)$.
   - InsertionSort make no more than $O(n\log(n)))$ pairwise comparisons.
   - SelectionSort is stable.

   SOLUTION:
   true, true, false, false, true
   END SOLUTION

2. Your friend shows you the following algorithm called `weiredSort` for sorting an array of `numbers`. He claims that it makes $O(n\log(n))$ comparisons in the worst case to sort `numbers`, since it is a Divide-and-Conquer algorithm. Of course, he is wrong. Explain why the number of comparisons is greater than $O(n\log(n))$.

5 pts.

```
void weiredSort(int[] numbers) {
        weiredSortRec(numbers, 0, numbers.length-1);
}

void weiredSortRec(int[] numbers, int lo, int hi) {
        if (hi - lo > 0) {
                int mid = (hi + lo) / 2;
                weiredSortRec(numbers, lo, mid);
                weiredSortRec(numbers, mid+1, hi);
                sortPart(numbers, lo, hi);
        }
}

void sortPart(int [] numbers, int lo, int hi) {
        for (int i = lo; i < hi; ++i) {
                for (int j = lo; j < hi-i; ++ j) {
                        if (numbers[j] > numbers[j+1]) {
                                swap(numbers, j, j+1);
                        }
                }
        }
}

void swap (int[] numbers, int x, int y) {
        int temp = numbers[x];
        numbers[x] = numbers[y];
        numbers[y] = temp;
}
```

SOLUTION:
Let n be the length of the array. Only consider the call to `sortPart(numbers, 0, n-1)` on the top level of recursion, i.e. at `weiredSortRec(int[] numbers, 0, n-1)`. Inside the two loops in `sortPart(numbers, 0, n-1)` the comparison gets called $(n-1)+(n-2)+...+1 = (n-1)*n/2 = 0.5*n^2 - 0.5*n$ times, which already is $O(n^2)$. This is already more than $O(n\log(n))$.
END SOLUTION

8 Concurrency and Threads

1. Select the answer below which *best* fits: Two threads each hold a resource that the other is requesting. This is an example of:

   2 pts.

   - Deadlock
   - Resource contention
   - Livelock
   - Race condition

   SOLUTION:
   deadlock
   END SOLUTION

2. Select the answer below which *best* fits: When a program's result relies upon the execution order of a program's threads, it is said to contain a:

   2 pts.

   - Deadlock
   - Timing bug
   - Livelock
   - Race condition

   SOLUTION:
   race condition
   END SOLUTION

3. Java contains built-in support for writing threaded programs. An example of this would be the (circle all that apply):

   2 pts.

   - "synchronized" keyword.
   - "for" loop.
   - "Thread" class.
   - "private" operator.

   SOLUTION:
   synchronized and Thread.
   END SOLUTION

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

   5 pts.

   - Threads cannot access objects that were created by a different thread.
   - When two threads simultaneously call the same method, one thread may overwrite the values of the local variables of that method from the other thread.
   - A Java program ends when the thread that executed main() terminates.

- If you run a Java program on a computer with 2 processors/cores, you can create at most 2 threads.
- One starts a Java thread by calling the method run().

SOLUTION:
false, false, false, false, false
END SOLUTION

5. The following code may or may not be correct. It was written by someone who is looking to make a counter class which is shareable between many threads. If it is correct, state why. If it is incorrect, fix it.

4 pts.

```
public class ShareableCounter
{
    private int i;

    public ShareableCounter()
    {
        i = 0;
    }

    public int inc()
    {
        i = i + 1;
        return i;
    }
}
```

SOLUTION:
insert “synchronized” into “public synchronized int inc()”. Or put a synchronized block for “this” around the “i=i+1”.
END SOLUTION

9 Induction and Asymptotic Complexity

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

   4 pts.

   SOLUTION:
   There exists $c \geq 0$ and $N \geq 0$ such that for all $n \geq N$ it holds that $f(n) \leq cg(n)$.
   END SOLUTION

2. The following is a recursive version of InsertionSort. Write down the recurrence relation that describes the number of write accesses to the array (i.e. `array[...]=...`) made in the worst case.

   5 pts.

```
public static void sort(int[] array, int n) {
        // sorts the first n elements of array
        if(n == 0) {
            return;
        }
        else {
            int tmp = array[n-1];
            sort(array,n-1);
            int j;
            for (j = n-1; (j > 0) && (array[j-1] > tmp); j--) {
                array[j] = array[j-1];
            }
            array[j] = tmp;
        }
        return;
}
```

   SOLUTION:
   $T(0) = 0$
   $T(n) = T(n-1) + n$
   END SOLUTION

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

   6 pts.

   $T(1) = 3$
   $T(n) = T(n-1) + 2n$

   SOLUTION:
   To show: $T(n) \leq cn^2$ for some fixed c and all $n > N$.
   Base Case ($k = 1$): Set $c = 3$, then $T(1) \leq 3 \cdot 1^2 = 3$.
   Inductive Hypothesis: $T(m) \leq 3m^2$ for all $m \leq k-1$
   Induction Step ($k \to k+1$): $T(k+1) = T(k) + 2k \leq 3k^2 + 2k \leq 3k^2 + 3k = 3k(k+1) \leq 3(k+1)^2$.

   END SOLUTION