Final Exam

CS2110 Spring 2014

May 12, 2014

	1	2	4	5	6	7	Total
Question	True False	Short Questions	Searching/ Sorting	Trees/ Heap	Graphs	Threads	Total
Max	20	19	15	15	20	11	100
Score							
Grader							

The exam is closed book and closed notes. Do not begin until instructed.

You have **150 minutes**. Good luck!

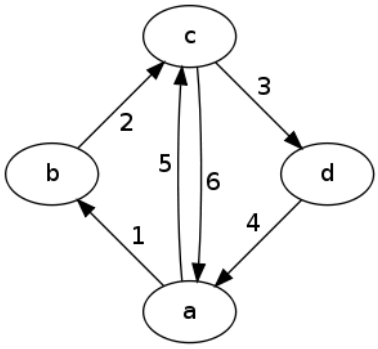
Start by writing your name and Cornell netid on top! There are 5 questions on 13 numbered pages, front and back. Check now that you have all the pages. When you hand in your exam, make sure your booklet is still stapled together. If not, please use our stapler to reattach all your pages!

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, so if something seems to need more space, you might want to skip that question, then come back to your answer and see if you really have it right.

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 itâĂŹs good style.

1. True / False [20 points]

a)	T	F	Consider hashing, as presented in recitation. If a value hashes to index k, quadratic probing (instead of linear probing) means looking at the values at indices k, $2k$, $3k$, $4k$ ... (all mod the table size, of course).
b)	T	F	The maximum number of threads cannot exceed the number of processors available on a computer.
c)	T	F	Dijkstra's algorithm is guaranteed to find the shortest path from the start node to every reachable node in the graph.
d)	T	F	Suppose class B is a subclass of class C. After the assignment $B\ ob =$ **new** $B()$; , ob **instance of** C is false.
e)	T	F	Even though Java has priorities for threads, a high-priority thread may get stuck waiting for a low-priority thread.
f)	T	F	The worst case running time of Quicksort is $O(n^2)$.
g)	T	F	Java has three kinds of variable: the field (instance variable), the parameter, and the local variable.
h)	T	F	Given the definition of the *median* of a population as *the value that is in the middle when the population is sorted*, we conclude that in a full and complete BST, the value at the root is the median.
i)	T	F	The following data structure is a Directed-Acyclic Graph (DAG). 
j)	T	F	In a quantum computer, when the output is observed, the wave function collapses to a classical state.

2. Short Questions [15 points]

2.a Parsing (4 points)

Below is a simple context-free grammar (ϵ denotes the empty string):

$$S \to BS \mid \epsilon$$
$$B \to \alpha B \beta \mid B\beta \mid \alpha\beta$$

Which of the following strings are legal candidates for S in this grammar? Put "Yes" in the brackets if it is legal, and put "No" if it is not.

[ ] $\alpha\alpha\beta\beta$

[ ] $\beta\beta\alpha$

[ ] α

[ ] $\alpha\beta\beta\alpha\beta\alpha\alpha\alpha\beta\beta\beta$

2.b Graphs (5 points)

A simplified automobile manufacturing process has the following subtasks, which may depend on one another.

A: Parts Purchase

B: Pressing Process

C: Welding Process

D: Tire Production

E: Body Manufacturing

F: Final Assembly

The dependence relations are shown below:

- Final Assembly depends on Parts Purchase, Tire Production, and Body Manufacturing.
- Body Manufacturing depends on Pressing Process and Welding Process
- Tire Production depends on Parts Purchase and Pressing Process

Draw a Directed Acyclic Graph (DAG) to represent these dependence relations, where an edge (u, v) indicates v depends on u.

2.c GUIs (6 points)

Look at the following GUI constructor:

```
public class GUI implements ActionListener {
  private final JFrame frame= new JFrame("Turing Award Winners");
  private final JTextArea textArea= new JTextArea();
  private JButton button = new JButton("2012-2013");

  public GUI() {
    button.addActionListener(this);
    frame.add(button, BorderLayout.NORTH);
    frame.add(textArea,BorderLayout.CENTER);
    textArea.setPreferredSize(new Dimension(400,200));
    frame.pack();
    frame.setVisible(true);
  }

  /** Called when button is pressed. */
  public void actionPerformed(ActionEvent e) {
    frame.setTitle("Turing Award Winners: 2012-2013");
    textArea.setText("2013: Leslie Lamport\n");
    textArea.append("2012: Silvio Micali, Shafi Goldwasser");
    frame.pack();
  }
}
```

(a) Draw the presented object when the constructor shown above is executed.

(b) Draw the object after the button "2012-2013" is clicked.

2.d Invariant (4 points)

The following code is to store in s the sum of b[0..n]. Complete the code. You must use the given loop invariant.

```
int k=             ;

s=  0;
//  invariant: -1 <= k <= n  &&
//              s is the sum of b[k+1..n]

while (                                  ) {

     s=                          ;


     k=                          ;


}
```

3. Searching / Sorting [15 points]

3.a Search (2 points)

a) What is the complexity of binary search on an array of size n:

b) For a linked list, the search time can be improved by keeping the list sorted. What is the time complexity of insertion to a sorted linked list in terms of the number of nodes n?

3.b Insertion Sort (1 point)

Write the invariant for `insertionSort` method.

```
/** Sort b[h..k] --put its elements in ascending order. */
public static void insertionSort(Comparable[] b, int h, int k) {
    // Invariant:


    ....
}
```

3.c Selection Sort (6 points)

An `Integer` array `b` holds values in the following order: 5, 19, 13, 3, 9, 11, 7, 17, 2
Complete the following table such that every row contains the state of `b` after the currently selected finds its place.

(Initial)	5	19	13	3	9	11	7	17	2
(Step1)									
(Step2)									
(Step3)									
(Step4)									
(Step5)									
(Step6)									
(Step7)									
(Step8)									

3.d Merge Sort (6 points)

The code for merge sort is given as follows:

```
/** Sort b[h..k]  --put its elements in ascending order. */
public static void mergeSort(Comparable[] b, int h, int k) {
    if (h >= k) return;

    int e= (h+k)/2;
    mergeSort(b, h, e);   // Sort b[h..e]
    mergeSort(b, e+1, k); // Sort b[e+1..k]
    merge(b, h, e, k);    // Merge the 2 segments
}
```

a) An `Integer` array `b` holds values in the following order: 5, 19, 13, 3, 9, 11, 7, 17, 2
If we call `mergeSort(b, 0, b.length-1)`, how many calls to method `merge` will be made in total?

b) Let's say the answer for part (a) is n. Complete the following table such that it represents the state of `b` just after the last 4 returning `merge` calls.

Step(n-3)									
Step(n-2)									
Step(n-1)									
Step(n-0)									

4. Heap [15 points]

4.a Insertion (5 points)

Draw the heap that results from inserting the following sequence of integers, in the order given, into an empty min-heap. (Hint: Your answer should be a binary tree.)

$$10, 5, 7, 2, 1, 30, 15, 9$$

It will help us give partial credit if you show the heap after each insertion.

4.b Extraction (10 points)

Implement the `bubbleDown` function according to the specification below.

```
public class MinHeap {
    int size;     // The heap is in array[0..size-1], stored as
    int[] array;  // described in the CS2110 class lectures/notes

    /** Remove and return the smallest element
     (throws an exception if heap is empty) */
    public int extract() {
        if (size == 0)
            throw new RuntimeException("Heap is empty");
        int temp= array[0];
        array[0] = array[size-1];
        size--;
        bubbleDown(0);
        return temp;
    }

    /** Bubble node k down to its heap position.
     *  It can be implemented either recursively
     *  or iteratively */
    private void bubbleDown(int k) {









    }
}
```

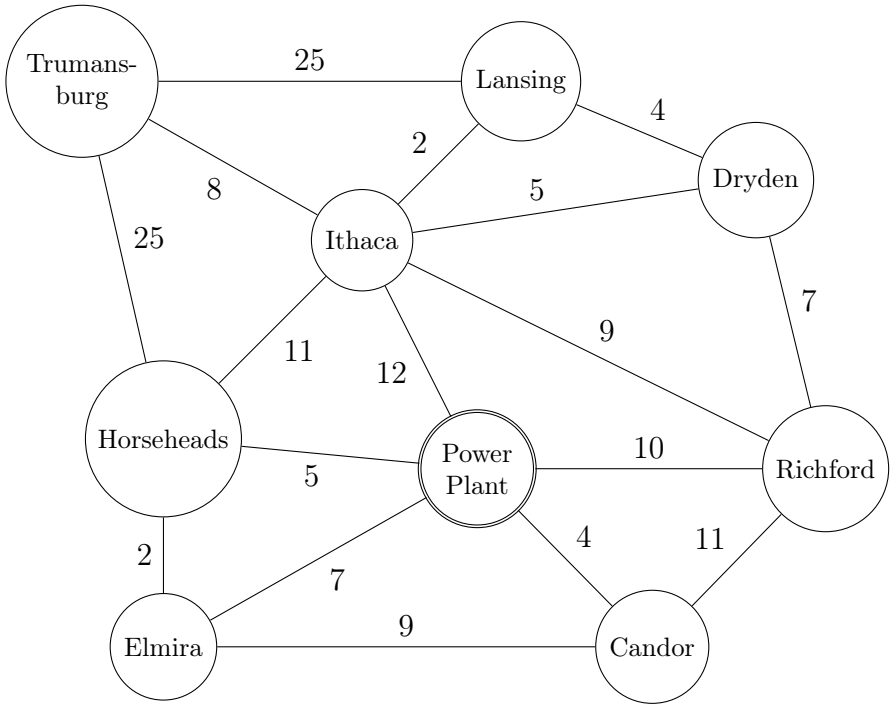
5. Graphs [20 points]

5.a Minimum spanning tree (10 points)

NYSEG has just constructed a new power plant in upstate New York, and needs your help determining how to connect it to the many small towns in the region. New power lines must be constructed to connect the towns' local grids to the power plant, but some regions are more difficult to build power lines through than others. Your job is to determine the least expensive way to connect all the towns into a network that includes the power plant.

The graph below shows the towns that need to be connected and the possible paths between them, along with the planning committee's estimate of the budget required to build power lines along each of those paths. The number annotating an edge represents the cost, in thousands of dollars, of building a high-tension power line along that edge. In order to find the lowest-cost set of power lines to build, you need to find the minimum spanning tree of this graph.

Starting at the node labeled "Power Plant," use Prim's algorithm to construct the minimum spanning tree of this graph. Show the order in which you add edges to the MST by labeling them with the letters A-Z, where A is the first edge you add, and leaving unlabeled edges you never add. Then circle the subset of the graph that is in the final MST.



5.b Depth First Search (10 points)

Fill in the method skeleton below to write the Depth-First Search algorithm.

```
/** An instance is a node of a graph */
public class GraphNode {
  public boolean visited = false; //true iff this node has been visited during dfs
  public GraphNode[] neighbors;  //neighbors of this node in the graph

  /* Other fields and methods omitted for brevity */
}

/** This node is unvisited. Visit all nodes reachable from this node (including
   * this node) along paths of unvisited nodes that start at this node. */
public void DFS() {









}
```

6. Threads [15 points]

A growing online retailer uses a queue implementation as a linked list to process orders from customers. The requests coming to the server are dispatched to multiple threads which manipulate the queue. Some part of the code is presented below.

```
public class OrdersQueue {
   // The orders are maintained in a singly linked list of objects of class OrderNode.
   // head points to the first order and tail to the last order.
   // head and tail are null if the queue is empty.
   private OrderNode head, tail;

   /** Append order to the queue */
   public void add(Order order) {
        if(tail != null) {
           tail.succ= new OrderNode(order, null);
           tail= tail.succ;
        } else {
           tail= new OrderNode(order, null);
           head= tail;
        }
   }

   /** If the queue is empty return null. Otherwise, remove the first order in the
       queue and return it */
   public Order poll() {
        OrderNode first= head;
        if(head != null) {
           head= head.succ;
           if(head == null) tail= null;
           return first.getOrder();
        } else
           return null;
   }

   /** An instance is a node of the list */
   private class OrderNode {
      private Order order;    // Information stored in the node.
      private OrderNode succ; // Successor of this node on the list.

      /** Constructor: an instance with successor s (s can be null),
       *  and order information o. */
      private OrderNode(Order o, OrderNode s) {
          order= o;
          succ=  s;
      }
      ...
   }
   ...
}
```

a) Define what a *race condition* is.

b) What might cause a race condition in `OrdersQueue` implementation? Suggest two solutions.