Name: ____________________________

CSE332 Summer 2010: Final Exam

Closed notes, closed book; calculator ok.

Read the instructions for each problem carefully before answering. Problems vary in point-values, difficulty and length, so you may want to work through the easier ones first in order to better budget your time. You have the full hour to complete this exam.

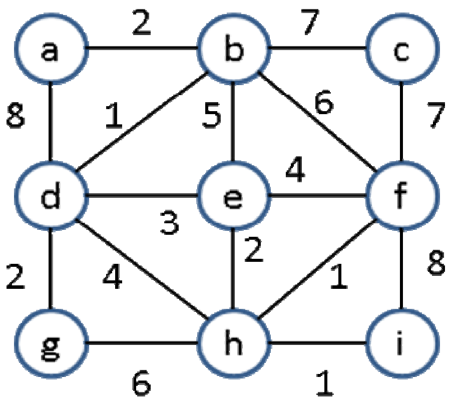
Good luck!

Problem	Max points	Score
1	12	
2	14	
3	16	
4	21	
5	21	
6	16	
Total	100	

Name: ____________________________

1.Dijkstra’s Algorithm

First, perform Dijkstra’s algorithm on the following graph to find the shortest path from vertex ‘**a’** to every other vertex. Break ties by choosing the lowest letter first; ex: if ‘a’ and ‘b’ were tied, you would explore ‘a’ first. Use the table below to show your work. Then, after completing the table, **write and circle the shortest path from ‘a’ to ‘i'.**



Vertex	Known	Cost	Parent
a			
b			
c			
d			
e			
f			
g			
h			
i			

2. Parallel Prefix Sum

Name: ______________________________

a. Given the following array as input, perform the parallel prefix sum algorithm, **using a sequential cutoff of 2**. Show your work by drawing the parallel tree, showing the following values for each node: **lo, hi, sum** and **fromleft**. Use the same tree for both steps of the parallel prefix – you do not need to draw 2 separate trees.

Input:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Element	**8**	**3**	**4**	**9**	**2**	**1**	**7**	**6**

b. How is the **fromleft** value computed for a node in the tree? Specifically, if you have a node with **sum** & **fromleft** computed, how do you compute **fromleft** for its left & right children (both of which have **sum** already computed).

Name: ___________________________

3. Amortized Bounds and Sorting

a. Recall that insertion in a binary heap is O(1) in the average case. **Disprove** an amortized bound of O(1) for insertion into a binary heap. Treat **M** (the number of inserts performed) as a variable in your answer. Recall that binary heap insertion works by placing the new element at the next position in the complete tree and percolating it up.

b. Imagine you work for a software company, and your boss asks you pick a good sorting algorithm for a program the team is working on. In this particular situation, run-time speed is critically important, but it's not a big deal if the algorithm runs slowly every now and then – it just needs to run fast for the most part. Memory use is an issue, however; the sort you choose shouldn't use any more memory than is necessary. What sorting algorithm would you choose? Justify your choice in a couple of sentences.

c. Recall that comparison-based sorting has a bound of Ω(nlogn) for the worst case. In a sentence or two, describe how the lower bound on comparison-based sorting was proven. You need not show any arithmetic or even the particular constant used for the exponential in the proof, but you should provide enough detail to convince the reader of the idea.

4. Concurrency

Consider a graph library in Java that provides an API for following edges and for reading and adding-to an integer field stored at each vertex. The library allows concurrent access via a fine-grained locking scheme. In this scheme, each vertex has its own re-entrant lock, and in order to modify a series of vertices on a path, the library acquires each lock, starting with the first in the path, then the second, etc. until each stop on the path has been iterated through and has its lock acquired. Only after the computations regarding the path are complete are all of the acquired locks released. Using these locks, data races on vertex data are entirely prevented.

a. What *specific* concurrency problem can this scheme fall victim to? Briefly describe how this could happen.

b. How could the problem in (a) be prevented, while still using fine-grained locking? Assume you can make changes to the library code to prevent the problem.

c. The problem in parts (a) & (b) aside, briefly describe an error that could occur within a **single thread** if the locks were **not** re-entrant.

Name: ______________________________

5. Concurrency II

Consider the code below:

```
public class StringArray {
     int len=0;
     String[] arr=new String[4];
     synchronized void checkAndResize(){
     //checks to see if array is full; if so, doubles size and copies over
     //if not, returns (nothing changes)
     }
     synchronized int getLen(){return len;}
     synchronized void incLen(){len++;}
     synchronized void set(int index,String s){arr[index]=s;}

     void add(String s){
          checkAndResize();
          int myLength=getLen();
          set(myLength,s);
          incLen();
     }
}
```

a. Does this code have a data-race? Explain briefly.

b. Show an interleaving of two threads, both calling add() on the same StringArray that results in one added String being lost, with null being in its place instead.

c. Show an interleaving of two threads, both calling add() on the same StringArray, that results in an array out of bounds exception.

d.

6. Parallelization

In Java using the ForkJoin Framework, write pseudo-code to solve the following problem:
• Input: A Comparable[]
• Output: A Boolean value: **true** if the elements of the array are in **ascending** order, **false** if not.

Your solution should have O(n) work and O(log n) span where n is the array length. Do not employ a sequential cut-off: the base case should process one element.
Recall that the `Comparable` interface has a single method, `int compareTo(Comparable c)` which returns:

- 0 if the elements are equal
- A positive number if 'this' is larger than the element c
- A negative number if 'this' is smaller than the element c

You only need to give one class definition, `SortedCheck`; we have provided some of the code for you. Note: the given code requires that `fjPool.invoke`, when called on a `SortedCheck` instance, returns a Boolean value, so you'll need to write your class with that in mind.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;
class Main{
     static final ForkJoinPool fjPool = new ForkJoinPool();
     boolean isSorted(Comparable[] array) {
       return fjPool.invoke(new SortedCheck(array,0,array.length));
     }
}
```

Name: ____________________________

Extra credit

1. How is my (the instructor's) name (both first and last names) spelled? Just write it out. Worth 1 pt.

2. Prove that P!=NP. Worth 1 pt. Note: You'll get credit for anything you write down; that is, as long as this space isn't left blank, you'll get 1 pt.