NAME: _____ NETID: _____

CS2110 Fall 2009 Final Exam
December 16, 2009

Write your name and Cornell netid . There are 5 questions on 10 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck!

	1	2	3	4	5	Total
Score	/20	/20	/20	/25	/15	
Grader						

1. (20 points) Here's code for Quicksort. The code was on our web site and implements exactly the algorithm we discussed in class; it hasn't been changed and it works exactly the way it should.

```
public class QuickSort<T extends Comparable<T>> implements Sorter<T> {

   static Random rand = new Random();

   public void sort(T[] x) {
      sort(x, 0, x.length);
   }

   // sort the portion of Comparable array x between
   // lo (inclusive) and hi (exclusive) in place
   // does not touch other parts of x
   private void sort(T[] x, int lo, int hi) {

      // base case
      if (hi <= lo + 1) return;

      // Pick the middle element as the pivot
      T pivot = x[(lo+hi)/2];

      // partition the elements about the pivot
      int i = lo;
      int j = hi - 1;
      while (true) {
         // advance indices
         while (i < j && x[i].compareTo(pivot) < 0) i++;
         while (i < j && x[j].compareTo(pivot) > 0) j--;
         // done?
         if (i == j) break;
         // swap
         T tmp = x[i];
         x[i] = x[j];
         x[j] = tmp;
      }

      // recursively sort the partition elements
      sort(x, lo, j);
      sort(x, j, hi);
   }
}
```

(a) 10 points. In class we analyzed the complexity of this algorithm and showed that the typical complexity is O(n log n). But we also discussed a "worst case" complexity bound. Explain what conditions provoke the worst-case complexity, and what that complexity is. *You must explain your answer.*

Worst-case complexity arises when, rather than evenly dividing the list, only a constant number of elements falls into one of the partitions, on every level of the recursion (if, for instance, the pivot is always chosen to be the minimum or maximum). The result is $O(n^2)$ complexity.

Hint: For parts (b) and (c), the specific way of picking the pivot element matters. In the version of Quicksort given above, this is the line that selects the pivot: T pivot = x[(lo+hi)/2];

(b) 5 points. What would be the complexity of this version of Quicksort if it runs *only* on pre-sorted arrays (arrays that are already in the correct sorted order). Briefly explain your answer: *a correct answer without a good explanation will lose points*.

O(n log n) because the median will always be selected as the pivot.

(c) 5 points. How would you construct inputs that cause Quicksort to exhibit the worst-case complexity that you gave us in part (a).

We need to construct a vector in which the pivot value picked will always be the minimal value for the portion of the vector being sorted. For example, suppose our vector will contain the integers 0..n-1. We want the smallest value (namely, 0) to be in location (n-1)/2. Quicksort will pick 0 as its pivot on the first step, shifting 0 to the extreme left and leaving the remaining values (1..n-1) in the remaining cells (cells 1..n-1). Now we want 1 to be located in cell (1+n-1)/2, and so forth.

This makes Quicksort work very hard: each time it pivots, it does a comparison on all the values in the vector (on average, n/2 of them) and the vector only gets smaller by 1 (hence on average will be of length n/2). The complexity is thus $O(n^2)$

What makes this slightly tricky to actually "do" is that when Quicksort does the pivot operation, elements to the left of the pivot get shifted one cell to the right. For example, suppose n=5, and think of the input vector as a list of five variables: [a b c d e]. Quicksort will pick c as its pivot, so set c=0: [a b 0 d e]. After pivoting the first time, Quicksort will have [0 a b d e] and will use b as its pivot, so set b=1: [a 1 0 d e]. After the second pivot operation, we have [0 1 a d e] and Quicksort picks d as its pivot, so set d=2. The third pivot gives us [0 1 2 a e], so we want a=3 and e=4. Thus for n=5 the input should be [3 1 0 2 5].

It is possible to write out a simple nest loop that will fill in a worst-case vector of length n, but we aren't expecting people to do that for this exam.

2. (20 points) True or false?

a	T	F	In Java, it is possible to write code that can only be type-checked when the program is actually running.
b	T	F	If two classes **A** and **B** implement the same interface **I**, then **A** is a subclass of **B**, or **B** is a subclass of **A**.
c	T	F	If String **s1 = "Cleopatra",** and **String s2 = "Cleo"+"patra",** then **s1.equals(s2)** returns *true*.
d	T	F	In Java, it is perfectly legal to have multiple definitions for the same method that have different types of arguments.
e	T	F	If myArray is of type **Array<T>,** then the elements of **myArray all** have the same dynamic type.
f	T	F	Suppose class **foo** defines method m, subclass **foobar** extends **foo and** does not override m. Then if **fb** is a **foobar** object, **fb.m()** is legal and will runs **m** as defined in class **foo.**
g	T	F	Given **int a = something**, and **double b = something else**, then ((double) a < b) and (a < (int) b) are equivalent expressions.
h	T	F	If **B** is a subclass of **A** and **b** is an instance of **B**, then **((B)(A)b == b)** is true.
i	T	F	If an application requires O(1) cost for a particular method, that property would be specified as part of the type signature and will then be automatically checked by Java.
j	T	F	If any thread terminates by returning from its top-level method, the program in which that thread was running will terminate too, even if other threads were still active.
k	T	F	Java uses **synchronize, wait** and **notify** as tools for threads to coordinate their actions.
l	T	F	When using the predefined Java **HashMap<T1,T2>,** the programmer must supply two types, one describing the type of the key and one describing the type of the value.
m	T	F	If **foo** is of type **HashMap<Animal,ArrayList<Gene>>** then one might use **foo** to store and retrieve the list of genes associated with some animal.
n	T	F	If a recursive method lacks a base case, Java will discover this when you try to compile the method and will force you to fix the code before it can be executed.
o	T	F	A race condition arises if two or more threads share (reading and writing) an object without synchronization.
p	T	F	A deadlock can only occur if two or more threads end up waiting for one-another in a "cycle".
q	T	F	An exception handler is a special block of code that runs when some an exception of a particular class is thrown. It must fix whatever caused the exception, because when it completes, the operation that caused the exception will be retried.
r	T	F	If a class includes two methods with the same name but different type signatures, and one method calls the other, we would say that the method is recursive.
s	T	F	In class we discussed the use of induction to prove things about concurrently executing threads. In particular, using induction, we can prove that a traffic circle in which cars on the circle have priority over cars trying to enter will not be subject to deadlock.
t	T	F	The role of induction in a proof is analogous to the role of recursion in an algorithm.

3. (20 points) Suppose you are given a binary search tree class.

```
public class BST<T extends Comparable<T>> {

   T            nodeValue;   // Value for this node
   BST<T>       left;        // Left child, or null if none
   BST<T>       right;       // Right child, or null if none

   … usual code for methods Add and Lookup …
}
```

a) (10 points) Write a new method **public int size()** that is called on a BST node, and counts the number of nodes in the subtree rooted at "this". Don't add extra class variables, but you can use additional helper methods, or local variables, as needed. *Example: if size is called on a leaf, it returns 1; if called on the root of a tree, size returns the number of nodes in the entire tree.*

```
public int size()
{
        int tmp = 1;
        if(left != null) tmp += left.size();
        if(right != null) tmp+= right.size();
        return tmp;
}
```

b) (10 points) Write a method **public List<T> toList()**that is called on a **BST** node, and returns a list of node values for the subtree rooted at "this", in sorted order. Again, don't add class variables, but helper methods and local variables are fine. *Hint: It is not necessary to implement a sorting algorithm!*

```
public List<T> toList()
{
        List<T> l = new List<T>();
        addtoList(l);
        return l;
}
public void addtoList(List<T> l)
{
        if(left != null) left.addtoList(l);
        l.add(value);
        if(right != null) right.addtoList(l);
}
```

4. (25 points) Here's a map for part of the Paris subway system. Assume that the full map is represented as a directed graph and that each edge is labeled by how long the train takes. For example, Concorde-Invalides normally takes 2min 10secs.

a) (15 points) In class we learned about Dijkstra's shortest path algorithm. Suppose that you were designing the computer program to recommend routes. The program knows where the user is currently located (the station where he/she is posing the question), and where that person wants to go. How could you apply Dijkstra's algorithm to solve this problem? How does that algorithm work, and why does it solve our problem? Make sure to tell us how we would obtain the actual route to follow!

Given a starting node, Dijkstra's performs a breadth-first search, visiting first the start node, then its children, then their children, etc, and tracking the path length (so far) to each node; each time it discovers a shorter route to some node, it updates the distance estimate. For our purposes we would modify Dijkstra's slightly and also keep track of the path to node – the path that gave this shortest length. Each time we discover a better path, we would just replace the previous best path with the new one.

We only need to run Dijkstra's once for each starting location. When a traveler asks how to get from, say, Etoile to Notre Dame, we would just read through this path checking to see if any edges on the path involve switching trains. Then we can print the recommendation: Take the number 1 train in the direction of St. Chapelle for 4 stops. At Odeon, switch to the number 4 train in the direction of Auteil and take it for two more stops. Your trip is expected to take take 16min 18secs.

b) (10 points) Our map (and hence our graph) lacks any information about the time needed to switch trains. Suppose we are given a correspondence delays table, as shown to the right. Describe a simple way of modifying the graph representing the metro map, that lets us continue to use the same algorithm as in (a) but will now take correspondence delays into account. *Hint: your solution will add some extra nodes and some extra edges.* Make sure to tell us which nodes to add and which edges to add and what weight to put on the edges. Notice that the table isn't "symmetric" (this is because trains are more frequent on some lines).

Station	From	To	Avg delay
Champs Elysées	Yellow	Blue	12 mins
Champs Elysées	Blue	Yellow	9 mins
Concorde	Yelllow	Pink	7 mins
… etc			

An easy solution is to just add extra nodes so that a correspondence station is represented by one node for each train that stops there. Then we can add edges to represent the correspondence table and run our version of Dijkstra's algorithm on the resulting directed graph. The estimated travel times will now take into account the time needed to switch trains.

5. (10 points) You developed your genome phylogeny application as a single-threaded application, even though you own a 4-core computer. It needed 10 minutes to compute a Phylogeny graph.

Using a debugging tool you discover that the program is spending 8 minutes just reading the data files. The number of files is the same as in cs2110 (40), but the files themselves are much, much larger: the DNA contains billions of codons. In contrast, DNA in our cs2110 files had about 50,000 codons.

To fully leverage your computer, you decide to use threads:

1. The main thread first creates 40 threads, one for each Animal.
2. These threads read the Animal data files in parallel; the main thread waits for them to finish.
3. As each thread reads the file, it parses the animal's DNA, storing each parsed gene into a TreeSet<Gene> collection. This TreeSet is shared, hence the threads use *synchronize* when accessing it.
4. When finished, the Animal-file-reader threads terminate and the main thread then resumes and finishes the computation.

Your code only uses synchronization statements for step 3, and has no bugs or race conditions.

You were hoping to see the 8 minutes drop to 2, and hence for your whole program to run in about 4 minutes. However, it actually runs a little slower with threads than it originally did without them!

a) (5 points). Your kid brother comments out all the synchronize() statements and now your program finishes quite a bit faster than the original 8 minutes. What does this tell you about why it was running so slowly? Was your brother's idea a good one? *Note: zero credit for "Yes" or "No" as an answer. We want to know <u>why</u>!*

My brother's idea was terrible, but it does explain why the program was slow. Clearly, what was slowing things down was the locking being done to implement synchronization: the 40 threads are apparently spending so much time fighting to lock the TreeSet that they end up running one by one, and the overheads associated with synchronization are turning out be huge (it takes time to lock and unlock a shared variable). Probably the garbage collector is also busy because of the overheads associated with the extra memory being consumed by all these concurrently active threads.

Kid brother's idea was dreadful because I had added those synchronization locks to avoid concurrency problems such as race conditions. Sure, the program speeds up without locking, but it probably is quite buggy now – giving wrong answers, crashing sometimes, etc.

b) (5 points). Now we want you to generalize what you told us in part (a). In general, suppose that two concurrently active threads both execute synchronize(something) { … }. Describe some conditions under which the synchronize statement can *safely* be removed.

In general, synchronization locks are needed (only) when two or more threads share objects that some of the threads update. Locks can be avoided if the threads are only reading the shared object, and once we know that competing (concurrent) threads are blocked by some single lock, no additional locks are needed. So there may actually be many locks that can be eliminated without risk; perhaps I can get back some of the performance I was after. Indeed, maybe I can get a speedup. But eliminating synchronization without thinking twice, like my kid brother did, is just a guarantee of disaster!

c) (5 points). Now take (a) and (b) into account and tell, briefly, how to modify the solution so that it would actually speed up from multi-core parallelism. *Hint: for full credit, your proposal won't change the number of threads but will instead focus on the problem causing the slowdown.*

One solution might be to have each thread create a separate list of genes while it runs (this seems to be the slow "step"). Since the threads aren't sharing anything they won't need any locking at all. Then as each thread finishes up, it can lock the TreeSet and add the genes that it found. Hopefully, we'll gain a speedup during the period when no locks are needed at all, and the last step, of merging the results, won't take very long. In fact is certainly shouldn't take very long: in particular, that step shouldn't run slower than it did in the original program. So in principle, we can get most of our multicore speedup this way.