CMPS 256 — ADVANCED ALGORITHMS AND DATA STRUCTURES
Summer 2013 – 14
Final Examination
Monday July 14, Bliss 105, 8:00 – 10:00 a.m.

Instructions.
This quiz is scored out of 100.
This quiz is open book and open notes: you can use the textbook, notes you have taken in class, and your homework solutions.
Show your work, as partial credit will be given.
You may use any algorithm that was covered in class. Just give the name of the algorithm and the chapter or page number in the book where the algorithm is given. If taken from your lecture notes, just say "lecture notes."
For questions that require you to write an algorithm, you must **prove** that your algorithm has the required running time.

Please draw a horizontal line to separate each answer from the next.
Best of luck!

Problem 1 (20 points).

Give the running time of the following, expressed only in terms of n. You may express this as $\Theta(f(n))$, for suitable $f(n)$.

```
{$n > 0$}
$i := 1$;
while $i \leq n^3$
    $j := 1$;
    while $j \leq i$
        $j := j + 1$;
    endwhile
    $i := i + 1$;
endwhile
```

Sample solution. Running time given by the following summation, by putting the limits of the loop as the limits of the sum:

$$\sum_{i=1}^{n^3} \sum_{j=1}^{i} 1$$

the summand is 1 since inner loop body has constant running time. We could also have used a constant c. The cost of $i := i + 1$ is ignored since its lower order.

The above simplifies to

$$\sum_{i=1}^{n^3} i$$

which is $\Theta(n^6)$.

Problem 2 (20 points).

Part a (10 points). Given an LLRB, suppose that we swap the colors of all the edges. Is the result, in general, a valid LLRB? Explain.

Sample solution. No, can have red-red violations

Part b (10 points). Is it possible to have a LLRB with $1,000,000$ nodes and no red edges? Explain.

Sample solution. An LLRB with no red links must be compete, due to the black balance condition, and so it must have $2^n - 1$ nodes for some n. $1,000,000$ is not of this form, so an LLRB with $1,000,000$ nodes must have some red links.

Problem 3 (20 points).

Part a (10 points). Give an algorithm to count the number of nodes in each connected component of an undirected graph. Must run in time $O(V + E)$.

Sample solution. Run the conected components algortihm in the slides. Create a `counts` array, which is indexed by the component number. Traverse the `cc` array, and increment `counts[i]` for each node `v` with `cc[v] = i`.

Part b (10 points). Give an algorithm to determine if a connected undirected graph G contains an articulation point. An articulation point is a node whose removal would break G into two disconnected components.

Must run in time $O(V(V + E))$.

Sample solution. For each node v, do the following. Delete v and run the connected components algorithm and count the number of components. If count $= 2$ then v is an articulation point.

Problem 4 (20 points).

Let G be a weighted directed graph and suppose that G contains an edge from u to v swth weight 0. Suppose now that we merge the nodes u and v. Will the shortest path lengths in G remain the same, in general, or will they change. Explain.

Sample solution. No, since this can create new paths in G.

Problem 5 (20 points).

```
public class Merge
{
   private static Comparable[] aux;  // auxiliary array for merges

   public static void sort(Comparable[] a)
   {
      aux = new Comparable[a.length]; // Allocate space just once.
      sort(a, 0, a.length - 1);
   }

   private static void sort(Comparable[] a, int lo, int hi)
   {  // Sort a[lo..hi].
      if (hi <= lo) return;
      int mid = lo + (hi - lo)/2;
      sort(a, lo, mid);
      // Sort left half.
      sort(a, mid+1, hi);
      // Sort right half.
      if (less(a[mid+1], a[mid]) merge(a, lo, mid, hi); // MERGE ONLY IF a[mid] > a[mid+1]
   }
}

public static void merge(Comparable[] a, int lo, int mid, int hi)
{ // Merge a[lo..mid] with a[mid+1..hi].
   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)
   aux[k] = a[k];


   for (int k = lo; k <= hi; k++)   // Copy a[lo..hi] to aux[lo..hi].
      aux[k] = a[k];

   for (int k = lo; k <= hi; k++) {
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi)               a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
}
```

The version of mergesort shown above skips the call on `merge()` whenever `a[mid] <= a[mid+1]`. Let a be an array of size N and such that $a[i] = i$ for all $i = 0, \dots, N-1$. What is the running time of `sort(Comparable[] a)`? Prove your result by giving a recurrence for the running time and solving the recurrence. i.e., Θ.

Sample solution. Linear time. Since a is initially sorted, `merge` is never called. Hence the recurrence for the running time is $T(n) = 2T(n/2) + c$. Solution is $\Theta(n)$.