Name: __________________________________________

Email address (UWNetID): ________________________________

CSE 332 Winter 2015: Final Exam
(closed book, closed notes, no calculators)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you write down, so show your work! Use only the data structures and algorithms we have discussed in class so far.

Note: For questions where you are drawing pictures, please circle your final answer.

You have 1 hour and 50 minutes, work quickly and good luck!

Total: Time: 1 hr and 50 minutes.

<table>
<thead>
<tr>
<th>Question</th>
<th>Max Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
1) [10 points total] Sorting: (Assume that array sun[] has indices: 0 to size-1)

SunnySort(int[] sun) {
    for (int i = 1; i < size; i++) {
        int j;
        int temp = sun[i];
        for (j = i; j > 0 && temp < sun[j-1]; j--) {
            sun[j] = sun[j-1];
        }
        sun[j] = temp;
    }
}

a) [2 points] This is actually a sort mentioned in class. What sort is this? **Insertion Sort**

b) [4 points] Describe the best and worst case data set for this sort. (If all cases behave similarly, please state that.) What is the big-O running time of those two data sets?

Best Case data set? Sorted order

Best Case running time? \(O(n)\)

Worst Case data set? Reverse Sorted order

Worst Case running time? \(O(n^2)\)

c) [2 points] Is it an in-place sort? Why or why not? (no credit without a reason or a definition of in-place, for partial credit define in-place sorting)

Yes - uses only constant extra space (variable temp)

d) [2 points] Is it a stable sort? Why or why not? (no credit without a reason or definition of stable, for partial credit define stable sorting)

Yes - only swaps if value being inserted is less than what is being compared with
2) [6 points total] Multiple Choice & Short Answer

a) [2 points] What is the running time of Dijkstra’s algorithm (assuming an adjacency list representation, and priority queues are used)?

\[ \Theta(V \log V + E \log V) \]

b) [2 points] Given a graph with |V| vertices and |E| edges, what is the space requirement (in big-O) for representing the graph. Pick your answers from the following:

- O(|E|)
- O(|V|)
- O(|V|^2)
- O(|E| + |V|)
- O(|E| * |V|)
- O(|E|^2 + |V|)
- O(|V|^2 + |E|)
- O(|V|^2 + |E|^2)
- O(|V|^2 * |E|^2)

As an adjacency list?

\[ O(V + E) \]

As an adjacency matrix?

\[ O(V^2) \]

c) [2 points] Which of the two graph representations is preferable for a sparse graph (in terms of using the smallest amount of space) (choose the best answer)?

Circle ONE (and only ONE answer):

i. An adjacency list, but only if there is only one edge for each vertex

ii. An adjacency list, but only if the number of edges is less than |V|

\[ \boxed{iii. An adjacency list} \]

iv. An adjacency matrix, but only if |V| is large

v. An adjacency matrix, but only if |E| is large

vi. An adjacency matrix.
3) [14 points total] Graphs and Dijkstra’s Algorithm
Use the following graph for this problem:

![Graph Diagram]

a) [2 points] List a valid topological ordering of the nodes in the graph above (if there are no valid orderings, state why not).

None Possible, there is a cycle (e.g. ABDA)

b) [5 points] Step through Dijkstra’s Algorithm to calculate the single source shortest path from A to every other vertex. You must show your steps in the table below for full credit. Show your steps by crossing through values that are replaced by a new value. Break ties by choosing the lowest letter first; ex. if B and C were tied, you would explore B first. Note that the next question asks you to recall what order vertices were declared known.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Known</th>
<th>Distance</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>T</td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>T</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
<td>43</td>
<td>A C</td>
</tr>
<tr>
<td>E</td>
<td>T</td>
<td>5</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>84</td>
<td>E D</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>87</td>
<td>D F</td>
</tr>
</tbody>
</table>
c) [1 point] In what order would Dijkstra’s algorithm mark each node as known?

ACDFEBG

d) [1 point] List the shortest path from A to G?

ACDFG

e) [2 points] For each modification to the graph on the previous page described below, indicate whether the modification would cause Dijkstra’s algorithm to fail to find a shortest path when starting at A. That is, running Dijkstra’s would find a shortest path from A to some vertex which was not the actual shortest path to that vertex. Assume each modification is applied independently to the original graph (they are not all combined). Circle one for each of i. thru ii.

i. Add an edge DC with weight of -4 o.k. Dijkstra’s would fail

ii. Change weight on edge AC to -2 o.k. Dijkstra’s would fail

f) [2 points] Give a Minimum Spanning Tree (MST) of the graph below by highlighting the edges that would be part of the MST.

![Graph Diagram]

(g) [1 point] Kruskal’s algorithm to find the minimum spanning tree made use of a data structure we only used for Kruskal’s called __Disjoint Set or Union/Find __________________.
4) [10 points] In Java using the ForkJoin Framework, write code to solve the following problem:

- **Input**: An array of ints (does not contain duplicates)
- **Output**: the *maximum* value and *its location* (index) in the Input array.

- Do not employ a sequential cut-off: the **base case should process one element**. (You can assume the input array will contain at least one element.)
- Give a class definition, **FindMax**, along with any other code or classes needed.
- We have provided some of the code for you, you should also fill in the _______ part.

```java
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

class Pair {
    // You are not required to use this class
    int a;
    int b;
    public Pair (int a, int b) {
        this.a = a;
        this.b = b;
    }
}

class Main{
    static final ForkJoinPool fjPool = new ForkJoinPool();
    _______ findMax (int[] array) { 
        return fjPool.invoke(new FindMax(______, array.length, array));
    }
}

Please fill in the two “______” above and write your code on the next page.
public class FindMax extends RecursiveTask<Pair> {
    int low;
    int high;
    int[] array;

    public FindMax(int low, int high, int[] array) {
        this.low = lo;
        this.high = hi;
        this.arr = array
    }

    protected Pair compute() {
        if (high - low < 2) {
            return new Pair(low, array[low]);
        } else {
            int mid = (low + high) / 2;
            FindMax left = new FindMax(low, mid, array);
            FindMax right = new FindMax(mid, high, array);
            left.fork();
            Pair rightResult = right.compute();
            Pair leftResult = left.join();
            if (leftResult.b > rightResult.b) {
                return leftResult;
            } else {
                return rightResult;
            }
        }
    }
}

4) Continued. Write your code on this page.
5) [6 points] Speedup

Given a program where 75% of it is parallelizable (and 25% of it must be run sequentially) what is the maximum speedup you would expect to get with 5 processors. Note: You must show your work for any credit. For full credit give your answer as a number or a simplified fraction (not a formula).

\[
\text{Speedup} = \frac{T_1}{T_p} = \frac{1}{S + \left(\frac{1 - S}{\frac{p}{5}}\right)} = \frac{1}{\frac{0.25 + (0.75)}{5}}
\]

\[
= \frac{1}{0.25 + 0.15} = \frac{10}{4} = \frac{5}{2} = 2.5
\]
6) [10 points] Parallel Prefix FindMax:
   a) Given the following array as input, perform the parallel prefix algorithm to fill the output array with the maximum value contained in all of the cells to the left (including the value contained in that cell) in the input array. Fill in the values for lo, hi, max, and fromLeft in the tree below. Do not use a sequential cutoff. You can assume that the array contains only positive integers. Note: This is findMax, NOT sum!

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

   Left child’s fromLeft:
   
   Right child’s fromLeft:
   
   \[
   \text{max (parent’s fromLeft, sibling’s max)}
   \]

   or
   
   \[
   \text{max (fromLeft, left, max)}
   \]
7) [19 points] The following class implements a dictionary storing (int key, E data) pairs as a “move to front” unsorted linked list. It assumes no duplicate keys will be inserted.

```java
public class MoveToFrontList<E> {
    private Node front = null;
    // Remove Node containing key from the list & return data
    // associated with key or null if key not found.
    synchronized E delete(int key){
        // Insert (key, data) at the front of the list.
    synchronized void insert(int key, E data){
        front = new Node(key, data, front);
    }
    // Find (key, data) and move to the front of the list.
    // Return data associated with key or null if key not found.
    E findAndMove(int key){
        E tempData = delete(key);
        if (tempData != null) insert(key, tempData);
        return tempData;
    }
}
```

a) Does the code above have (circle all that apply):

- potential for deadlock,
- a data race, ✔️ a race condition, ✔️ none of these

b) If possible, show (using code as done in class) an interleaving of two or more threads where a value that is in the list would not be found. If not possible, explain why not.

```
findAndMove(a) { /* finds a */
tempData ← a's Data //delete a
findAndMove(a) { /* Doesn't find a */
tempData = null; /* a not there! */
}

insert(a, tempData)
return tempData
```

c) If we change the findAndMove method to be synchronized, now, does the code above have (circle all that apply) (Note: synchronized uses re-entrant locks):

- potential for deadlock, ✔️ a data race, ✔️ a race condition, ✔️ none of these

d) Say we added another find method to this class that only finds values, but does not move them (does not modify the list). (See the code for find on the next page.) If all other methods are synchronized, but our new find method is not synchronized, does the code above plus this new find method have (circle all that apply):

- potential for deadlock, ✔️ a data race, ✔️ a race condition, ✔️ none of these
e) Say that instead of using synchronized at all, we used a readers/writer lock on the list. Modify the code below to use a RWLock. Draw arrows and label them to show where you are locking and unlocking (be very exact with your arrows). Use any reasonable names for the RWLock methods you use. You can assume that RWLock is re-entrant. (You can ignore the delete method itself – assume it “does the right thing”.)

```java
class MoveToFrontList<E> {
    private RWLock lk = new RWLock();
    private Node front = null;
    // Remove Node containing key from the list & return data
    // associated with key or null if key not found.
    E delete(int key){
        // Insert (key, data) at the front of the list.
        void insert(int key, E data){
            front = new Node(key, data, front);
        }
        // Find (key, data) and move to the front of the list.
        // Return data associated with key or null if key not found.
        E findAndMove(int key){
            E tempData = delete(key);
            if (tempData != null) insert(key, tempData);
            return tempData;
        }
        // Find (key, data) but does NOT move it.
        // Return data associated with key or null if key not found.
        E find(int key, E data){
            Node temp = front;
            while (temp != null) {
                if (temp.key == key) return temp.data;
                temp = temp.next;
            }
            return temp;
        }
    }
}
```

f) Would you expect using a single readers/writer lock as shown above to have better or worse or the same performance as using synchronized on all methods? Explain – be specific.

Better - Could have multiple finds in parallel

Worse or Same 
no more concurrency (but more locking + unlocking)

11 of 13
8) [12 points] Sorting

a) **Radix Sort**: Give a formula for the worst case big-O running time of radix sort. For full credit, your formula should include all of these variables:

max\_value – the values to be sorted range from 0 to max\_value
radix – the radix or base to be used in the sort
n – the number of values to be sorted

Answer: \( O\left(\log_{\text{radix}} \max\_value \cdot (n + \text{radix})\right) \)

b) **Quicksort**: Give the recurrence for each of the following: (Note: We are NOT asking for the closed form.)

- **Quicksort (parallel sort & parallel partition) – best case span**
  
  Answer: \( T(n) = O(\log n) + T\left(\frac{n}{2}\right) \)

- **Quicksort (parallel sort & parallel partition) – worst case span**
  
  Answer: \( T(n) = O(\log n) + T(n-1) \)

c) Give big-O runtimes of the following in terms of \( n \). For parallel sorts, give the span.

\[ O(n \log n) \quad \text{Mergesort (sequential) – worst case} \]
\[ O(\log^2 n) \quad \text{Quicksort (parallel sort & parallel partition) – best case span} \]
\[ O(n \log n) \quad \text{Quicksort (parallel sort & parallel partition) – worst case span} \]
\[ O(n \log n) \quad \text{Quicksort (sequential) – best case} \]
\[ O(n^2) \quad \text{Quicksort (sequential) – worst case} \]
9) [5 points] Amortized

a) An Insert operation on a Binary Search Tree (BST) has the following running times (give your answer in big-O, N is number of items in the BST):

<table>
<thead>
<tr>
<th>Type of Running Time</th>
<th>Big-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst case running time</td>
<td>O(N)</td>
</tr>
<tr>
<td>Average case running time</td>
<td>O(log N)</td>
</tr>
<tr>
<td>Amortized running time</td>
<td>O(N)</td>
</tr>
</tbody>
</table>

b) Your brilliant TAs have come up with a new dictionary called a CSE332-Tree that has the following characteristics: Worst case running time for a find operation is O(N), Amortized running time for a find operation is O(1). Running times for insert and delete are the same as for AVL trees.

You and your partner are implementing an application that does a lot of find operations. Your partner thinks you should use an AVL tree for your application. Considering only the runtime of find operations, give an argument for why you should use a CSE332-Tree instead of an AVL tree?

If our application can tolerate having an occasional find that takes O(N), and since our application does a lot of find operations, basing our choice on the amortized running time makes sense. We know that with a CSE332-Tree a sequence of M finds would take about O(M) time. This would be better than the O(M * log N) time that we would expect a sequence of AVL operations to take.

Considering only the runtime of find operations, describe a situation where you should use an AVL tree instead of a CSE332-Tree.

If your application cannot tolerate an occasional find operation that takes time O(N), then we should pick AVL trees whose worst case time is O(log N).

10) [8 points] P, NP, NP-Complete

a) “NP” stands for Nondeterministic Polynomial

b) What does it mean for a problem to be in NP?

Given a candidate solution, we can verify whether the solution is correct in polynomial time.

c) Give two examples of NP-Complete problems:

- Hamiltonian Circuit
- Traveling Salesman

d) What should you do if you determine the problem you are trying to solve is NP-complete, yet you still need to solve it?

Use an approximation algorithm that guarantees something close to optimal or a heuristic that seems to work well most of the time. You may also be able to recognize that your problem is actually more restricted than you first thought (e.g. a graph is always a tree) and find an optimal solution in reasonable time for that case.