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.

Question	Max Points	Score
1	10	
2	6	
3	14	
4	10	
5	6	
6	10	
7	19	
8	12	
9	5	
10	8	
Total	100	

Name: ______________________________

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?

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?

Best Case running time?

Worst Case data set?

Worst Case running time?

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)

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)

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)?

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

$O(|E|)$, $O(|V|)$, $O(|V|^2)$, $O(|E|^2)$, $O(|E| + |V|)$, $O(|E| * |V|)$, $O(|E|^2 + |V|)$, $O(|V|^2 + |E|)$, $O(|V|^2 + |E|^2)$, $O(|V|^2 * |E|^2)$.

As an adjacency list?

As an adjacency matrix?

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 **<u>ONE</u>** (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|

iii. An adjacency list

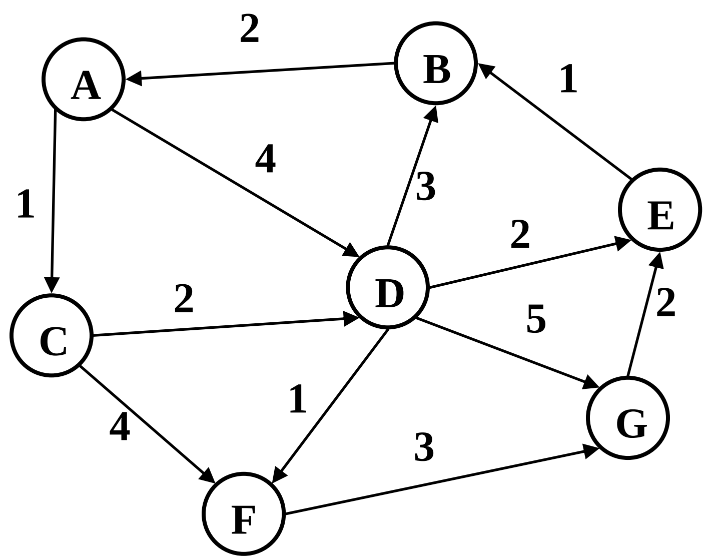
iv. An adjacency matrix, but only if |V| is large

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

vi. An adjacency matrix.

Name: _______________________________

3) [14 points total] Graphs and Dijkstra's Algorithm
Use the following graph for this problem:



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

b) **[5 points]** Step through Dijkstra's Algorithm to calculate the single source shortest path from A to every other vertex. You **<u>must show your steps</u>** 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*.

Vertex	Known	Distance	Path
A			
B			
C			
D			
E			
F			
G			

c) [1 point] In what order would Dijkstra's algorithm mark each node as *known*?

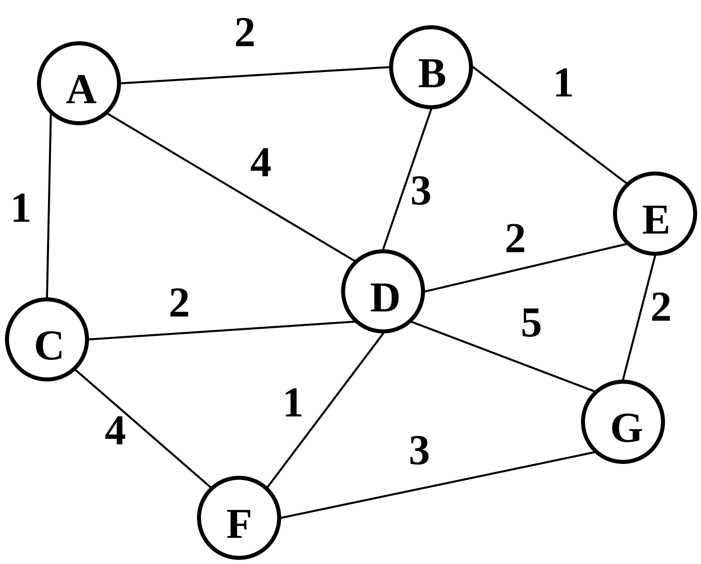
d) [1 point] List the shortest path from A to G?

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).

<u>Circle one for each of i. thru ii.</u>

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 <u>a</u> Minimum Spanning Tree (MST) of the graph below by highlighting the edges that would be part of the MST.



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 ____________________________________.

Name: ______________________________

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.

```
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(_____________________));
      }
}
```

Please fill in the two "_______" above and write your code on the next page.

Name: ______________________________

4) Continued. Write your code on this page.

Name: ______________________________

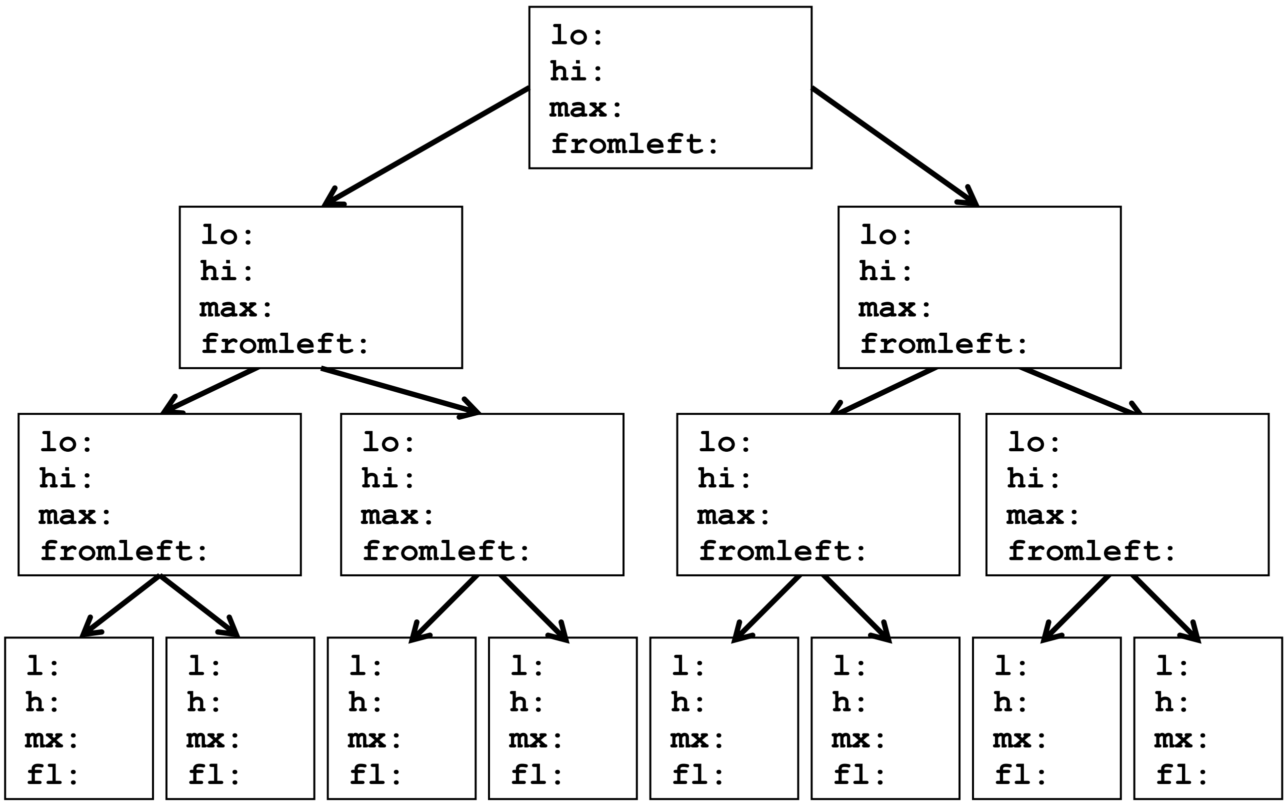
5) **[6 points] Speedup**

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: **<u>You must show your work for any credit</u>**. For full credit give your answer as a number or a simplified fraction (not a formula).

Name: ______________________________

6) **[10 points] Parallel Prefix <u>FindMax</u>:**

a) Given the following array as input, perform the parallel prefix algorithm to fill the **output** array with the **<u>maximum</u> 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!



Index	0	1	2	3	4	5	6	7
Input	**3**	**5**	**4**	**7**	**2**	**8**	**9**	**40**
Output								

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:

Name: ______________________________

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.

```
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.

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 <u>not</u> **`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

Name: ______________________________

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 <u>very exact</u> with your arrows).** Use any reasonable names for the `RWLock` methods you use. You can assume that `RWLock` is re-entrant. (<u>You can ignore the `delete` method itself</u> – assume it "does the right thing".)

```
public 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 ***<u>all</u>*** methods? **Explain** – be specific.

g) What if instead of using a single readers/writer lock on the whole list as shown above, you used a readers/writer lock on every *individual node*. We will require that all operations lock every node they touch in the list until they are done with their operation. `delete`, `insert`, and `findAndMove`, will lock all nodes in their path for writing, while `find` will lock all nodes in its path for reading. Would you expect this to have better or worse or the same performance as a single lock? **Explain** why – be specific.

Name: ______________________________

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:

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:

Quicksort (parallel sort & parallel partition) – worst case span

Answer:

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

________________ Mergesort (sequential) – worst case

________________ Quicksort (parallel sort & parallel partition) – best case span

________________ Quicksort (parallel sort & parallel partition) – worst case span

________________ Quicksort (sequential) – best case

________________ Quicksort (sequential) – worst case

Name: ______________________________

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):

Worst case running time ______________

Average case running time ______________

Amortized running time ______________

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*?

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

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

a) "NP" stands for ____

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

c) Give two examples of NP-Complete problems:

________________________________ and __________________________________

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?