

Name: Sample Solution

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? *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)?

$$O(V \log V + E \log V)$$

- 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:

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

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)?

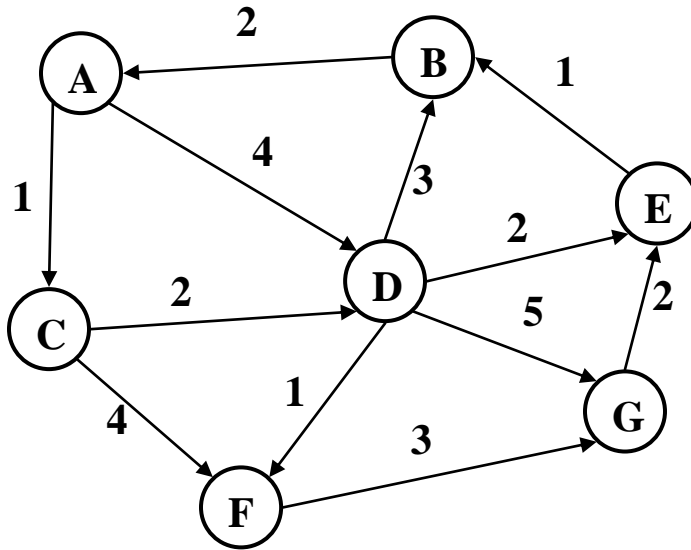
Circle **ONE** (and only ONE answer):

- 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 $|V|$
- 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).

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.

Vertex	Known	Distance	Path
A	T	0	
B	T	6	D
C	T	1	A
D	T	3	C
E	T	5	D
F	T	4	D
G	T	7	F

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

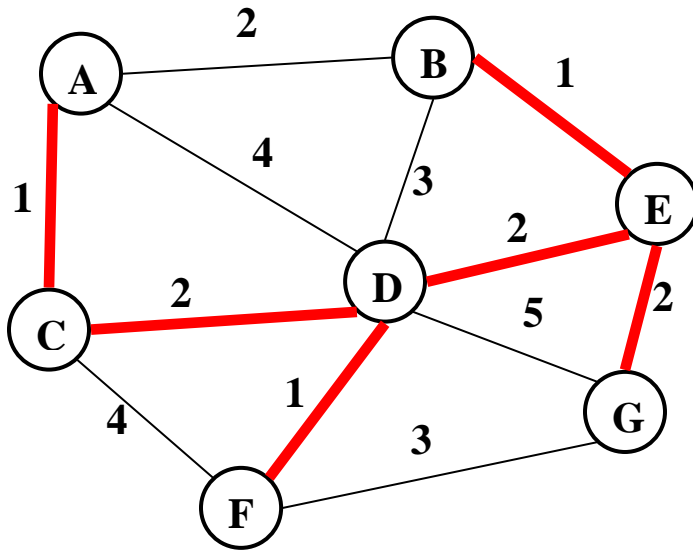
eg ADC = 0 vs AC = 1

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.



(can also select AB + leave out one of the other length 2 edges)

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.

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();
    _____ Pair findMax (int[] array) {
        return fjPool.invoke(new FindMax( 0, array.length, array ));
    }
}
```

Please fill in the two “_____” above and write your code on the next page.

4) **Continued.** Write your code on this 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;
            }
        }
    }
}
```

Name: _____

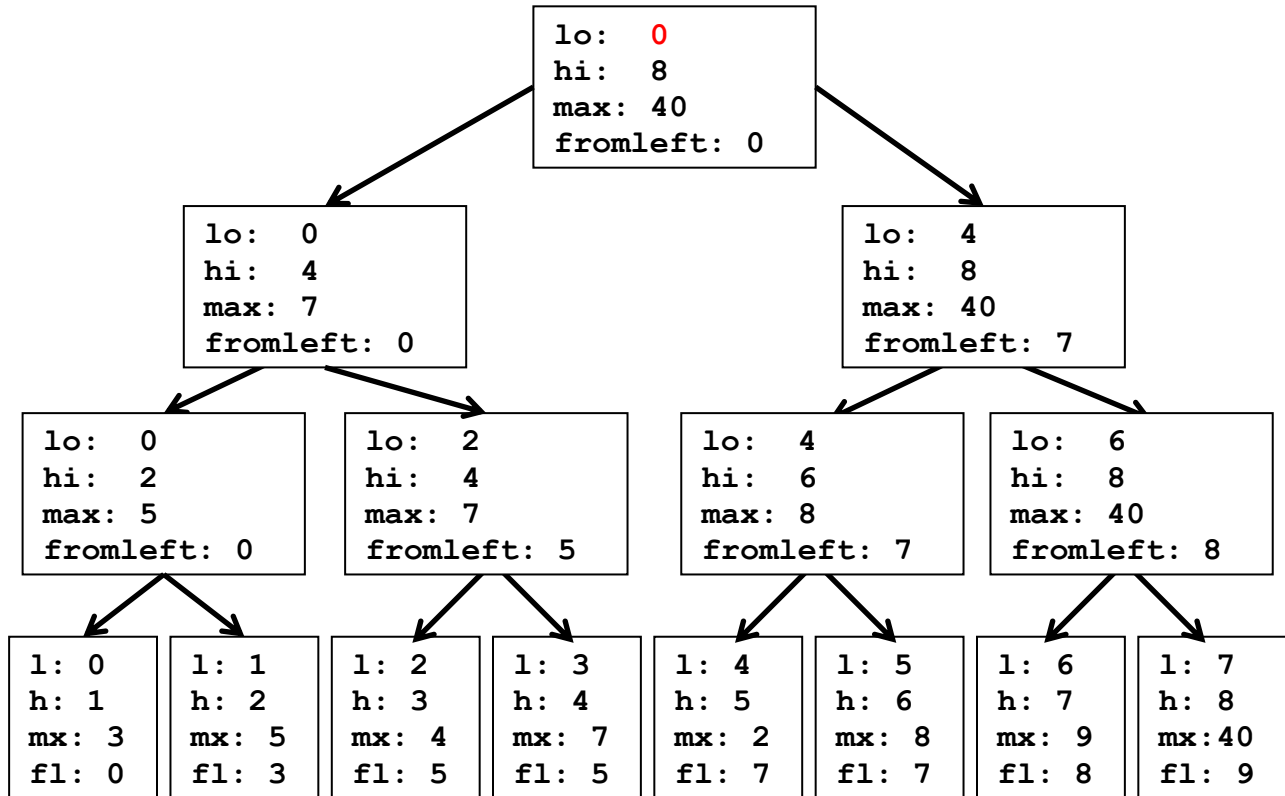
5) [6 points] Speedup

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

$$\begin{aligned} \text{Speedup on } P \text{ processors} &= \frac{T_1}{T_p} = \frac{1}{S + \left(\frac{1-S}{P}\right)} = \frac{1}{0.25 + \left(\frac{0.75}{5}\right)} \\ &= \frac{1}{0.25 + 0.15} \\ &= \frac{1}{0.40} = \frac{10}{4} \\ &= \frac{5}{2} = 2.5 \end{aligned}$$

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!



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

- 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:

parent's fromLeft or fromLeft

Right child's fromLeft:

*max (parent's fromLeft, sibling's max)
or max (fromLeft, left.max)*

Name: _____

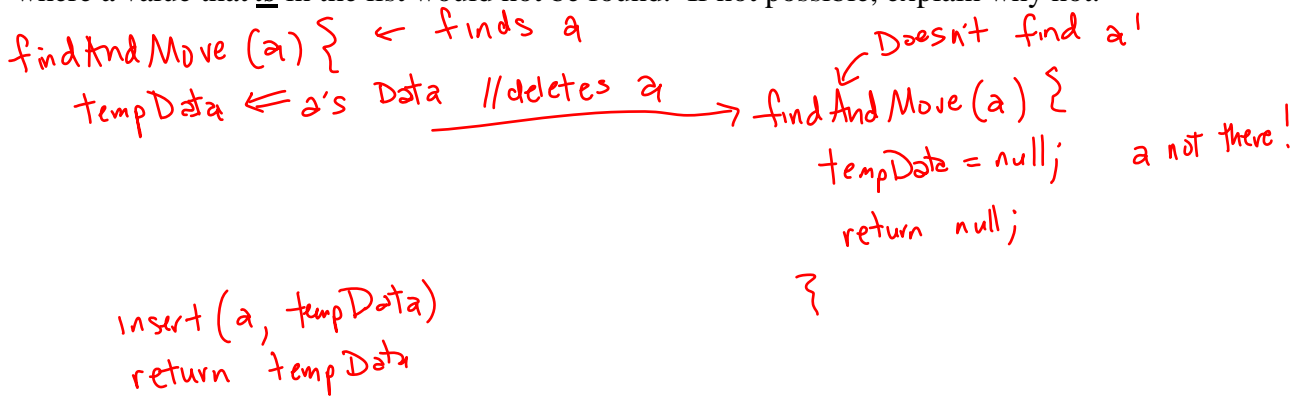
7) [19 points] The following class implements a dictionary storing (int key, E data) pairs as a “move to front” unsorted linked list. It assumes no duplicate keys will be inserted.

```
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 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

find reads front while someone else could be writing it

- 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”.)

```

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;
    }
}

```

Handwritten annotations in red:

- Arrows pointing to `lk.acquire-write();` and `lk.release-write();` for the `insert` method.
- Arrows pointing to `lk.acquire-write();` and `lk.release-write();` for the `findAndMove` method, with a note: "to prevent bad interleavings".
- Arrows pointing to `lk.acquire-read();` and `lk.release-read();` for the `find` method.

- 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

- 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.

Worse or Same → no more concurrency (but more locking + unlocking) (all must lock the front)

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:

$$O(\log_{\text{radix}}^{\text{max_value}} \cdot (n + \text{radix}))$$

- 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.

$O(n \log n)$ Mergesort (sequential) – worst case

$O(\log^2 n)$ Quicksort (parallel sort & parallel partition) – best case span

$O(n \log n)$ Quicksort (parallel sort & parallel partition) – worst case span

$O(n \log n)$ Quicksort (sequential) – best case

$O(n^2)$ 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):

Worst case running time _____ $O(N)$ _____

Average case running time _____ $O(\log N)$ _____

Amortized running time _____ $O(N)$ _____

- 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** _____ and _____ **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.