Name: _____

Email address (UWNetID): _____

CSE 332 Autumn 2019 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. Writing after time has been called will result in a loss of points on your exam.

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	Topic
1	9	Hashing
2	9	Graphs
3	6	More Graphs
4	10	|| Prefix
5	16	Concurrency
6	15	Sorting
7	10	P/NP
8	6	Speedup
9	14	ForkJoin
Total	95	

1) [9 points total] Hash Tables

a. **[3 pts]** Insert the following elements in this order: 37, 58, 19, 8, 17, 7 into the Double Hashing hash table below. You should use the primary hash function: `h(k) = k%10` and a secondary hash function: `g(k) = 1 + (k%6)`. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining elements.

0	1	2	3	4	5	6	7	8	9

b. **[2 pts]** The Quadratic Probing hash table below already contains several elements, including two that have been lazily deleted. Insert the elements 97 and 44 (in that order) into the table below. You should use the primary hash function: `h(k) = k%10`. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining elements.

0	1	2	3	4	5	6	7	8	9
10	(deleted)	21		14	(deleted)	16	37	48	

c. **[2 pts]** You are designing a separate chaining hash table and you are trying to decide what each bucket should point to. The options are using a linked list where items are inserted at the front of the list or an AVL tree. **List one positive point for each approach when compared to the other approach**. Be specific about how that data structure gives you that behavior.

Linked List (where items are inserted at the front of the list):

AVL Tree:

d. [2 pts] Give a tight big-O bound for the worst case runtime of a **Find** operation in a linear probing hash table **of size N^2** containing N elements. There have NOT been any deletions in the table. **Explain your what the worst case scenario is and why it would have this runtime.**

Worst Case Running time of Find:

2) [9 points total] Graphs!

a) **[4 pts]** You are given Kruskal's algorithm implemented using a priority queue and disjoint sets, as described in lecture. You are given a new implementation of disjoint sets with worst case running time of O(N) for find() and O(log N) for union(). Given that you must use this disjoint set implementation, what will be the worst case running time of Kruskal's? Give your answer as a tight Big-O bound in terms of V and E. **Explain how you got your answer briefly.**

Worst Case Running Time of Kruskal's with new disjoint sets:

b) [2 pts] What is the minimum number of edges in a **connected** undirected **graph with 6 vertices**? **Draw a picture of your graph.**

Minimum Number of Edges:

c) [2 pts] Give an EXACT number (**in terms of V**) for the minimum number of edges in a **complete** undirected graph without self-loops.

Minimum Number of Edges:

d) **[1 pts]** True/False: The MST for a graph contains the shortest weight path between every pair of vertices. If true, explain briefly. If false, give a counterexample. Circle one: **TRUE** or **FALSE**

3) [6 points total] More Graphs!

a) **[2 pt]** For each of the following situations, provide an appropriate graph algorithm for solving the problem:

i. Given a social network graph, where nodes are people and edges connect people who are friends, find how many degrees of separation you are from a given target person. "Degrees of separation" is the smallest number of people you need to go through to be connected to the target person.

Algorithm: _____

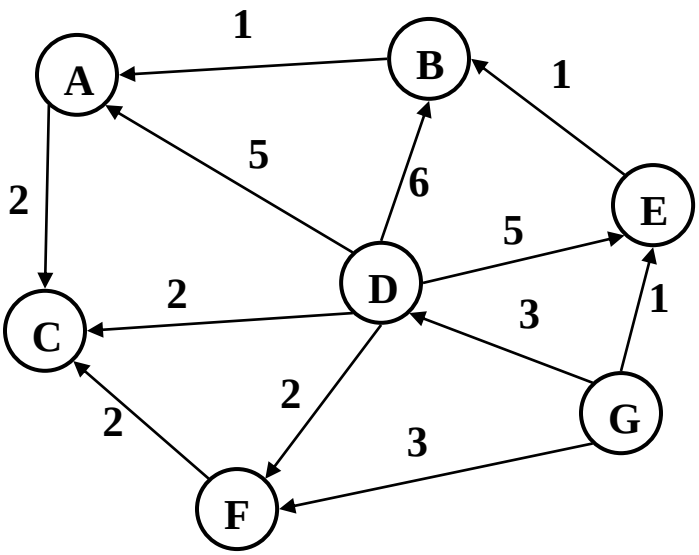
ii. Given a graph where the nodes are locations and the edges are the time needed to travel between locations, find the shortest path from the location "home" to the location "school".

Algorithm: _____

b) [2 pts] If you needed to calculate the **in-degree** of a single vertex in a graph, which representation would you prefer (circle one). Assume the basic representation of these we discussed in lecture:

adjacency matrix or **adjacency list**

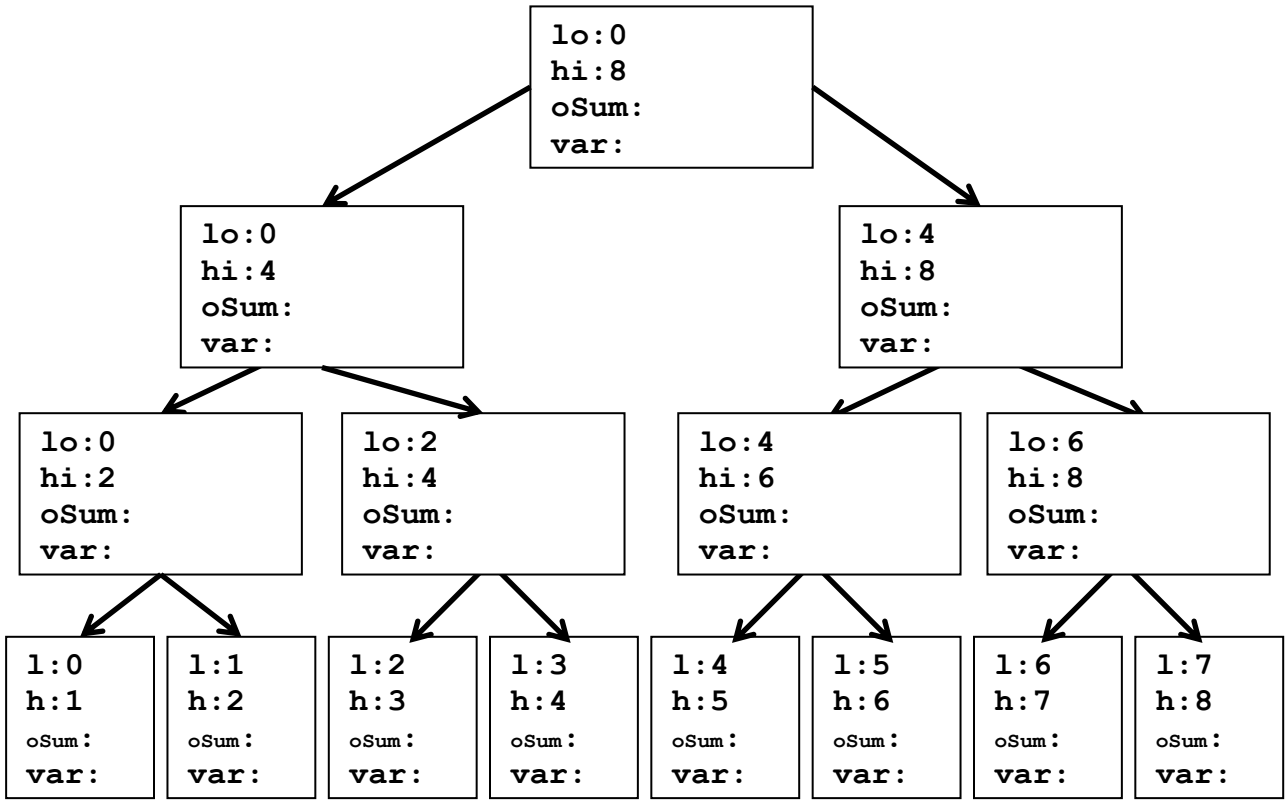
In a couple of sentences describe WHY?



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

4) [10 points] Parallel Prefix Sum of Odds:

a) Given the following array as input, perform the parallel prefix algorithm to fill the output array with the sum of only the odd values contained in all of the cells to the left (including the value contained in that cell) in the input array. Even values in the input array should not contribute to the sum (odd negative values should). Do not use a sequential cutoff.
For example, for input: {3, 14, -1, 4, 5, 5, 6, 1}, output should be: {3, 3, 2, 2, 7, 12, 12, 13}.
Fill in the values for **`oSum`** , **`var`** and the **`output`** array in the picture below.



Index	0	1	2	3	4	5	6	7
`input`	**-3**	**5**	**4**	**15**	**5**	**10**	**7**	**2**
`output`								

b) Give formulas for the following values where **`p`** is a reference to a non-leaf tree node and **`leaves[i]`** refers to the leaf node in the tree visible just above the corresponding location in the **`input`** and **`output`** arrays in the picture above.

`p.oSum` =

`p.right.var` =

`p.left.var` =

`output[i] =`

c) Describe how you assigned a value to **`leaves[i].oSum`** .

5**) [16 points total] Concurrency:** The `TimeMachine` class (code for entire class shown below) manages the CSE 332 staff's time machine. Multiple threads can access the same `TimeMachine` object.

```
1  public class TimeMachine {
2     private int now = 1985;
3     private int future = 2015;
4     private int energy = 100;
5
6
7
8
9     public boolean hasEnergy() {
10
11       return energy >= 100;
12
13    }
14
15    public void adjustEnergy(int charge) {
16
17       if (energy + charge < 0 ) { // energy should never be negative
18
19          return;
20       }
21       energy = energy + charge;
22
23    }
24
25    public void setFuture(int newFuture) {
26
27       future = newFuture;
28
29    }
30 }
```

a) **[4 pts]** Does the `TimeMachine` class as shown above have (circle **all** that apply):

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

Justify your answer. Refer to line numbers in your explanation. Be specific!

5) (Continued)

b) [4 pts] We now add this method to the `TimeMachine` class:

```
28  public boolean backToTheFuture() {
29
30      if (!hasEnergy() && now != future) {
31
32          return false;
33
34      }
35
36      now = future;
37
38      energy = energy - 100;
39
40      System.out.println("Heading to:" + future + " Energy remaining:" + energy);
41
42      return true;
43
44  }
```

Does this modified `TimeMachine` class have (circle **all** that apply):

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

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example. Refer to line numbers in your explanation. Be specific!

c) **[8 pts]** Modify the **code above in part b) and on the previous page** to use locks to ***allow the most concurrent access*** and to avoid all of the potential problems listed above. **For full credit you must allow the most concurrent access possible without introducing any errors or extra locks.** Create locks as needed. Use any reasonable names for the locking methods you call. **DO NOT use `synchronized`**. You should create re-entrant lock objects as follows:

```
ReentrantLock lock = new ReentrantLock();
```

6) [15 points total] Sorting

a) **[3 pts]** Give the ***recurrence*** for the PARALLEL MERGE discussed in lecture – **worst case span**. Note: We are NOT asking for the closed form. This is just the merge routine, *not full Mergesort*. **For any credit, explain all parts of your answer briefly**.

b) [4 pts] Give an exact expression in terms of N (closed form, not a recurrence, not Big-O) for the total number of times that **`quicksort`** and **`partition`** are called in a call to SEQUENTIAL QUICKSORT in the **best case**. Assume that partitioning is carried out by calling the helper method **`partition.`** Include the initial call to **`quicksort`** in your count. You should assume that N is a power of 2 and that no cutoff is used, **`quicksort`** will be called on a problem of size 1, **`partition`** will not be called on a problem of size 1. We want a simple expression without any summations. **Explain your answer.**

Exact Number of Times **`quicksort`** is called for problem of size N:

Exact Number of Times **`partition`** is called for problem of size N:

6**) (Continued)**

c) **[2 pts]** Give a tight Big-O bound for the running time of **insertion sort** if it *happened to be given an array of size N that was already sorted from largest to smallest*, and we wanted it sorted from smallest to largest. For any credit explain your answer.

Running time:

d) **[3 pts]** Is Sequential Quicksort sort in-place? Circle one: YES NO

Explain your answer. Be specific – **refer to the Sequential Quicksort algorithm in particular**, do not just give a definition of in-place. Be sure to mention all relevant parts of the quicksort algorithm.

e) **[3 pts]** Ruth is considering using a single bucket sort to sort all of the students currently registered for CSE 332 (200 students) by their student number. Would you recommend she do this?

Circle one: YES NO

Explain your answer.

7) **[10 points total] P, NP, NP-Complete**

a) **[1 pt]** "NP" stands for _____

b) **[2 pts]** Draw a diagram demonstrating how (we are pretty sure) the sets P, NP, and EXP overlap/don't overlap with each other. Label each set clearly.

c) **[2 pts]** What should you do if you **suspect** (but are not sure) a problem you are given is NP-complete, and **you need an answer to the problem in a reasonable amount of time**?

d) **[2 pts]** If someone finds a polynomial time algorithm to solve the Traveling Salesman problem, name **one thing** that would likely be true (about the running times of algorithms):

e) **[3 pts]** For the following problems, circle **ALL** the sets each problem belongs to:

Finding a minimum spanning tree in an undirected graph.	NP	P	NP-complete	None of these
Finding a path in a graph that begins and ends at the same vertex, and visits every vertex exactly once.	NP	P	NP-complete	None of these
Finding the shortest path from one vertex **to every other vertex** in a weighted directed graph.	NP	P	NP-complete	None of these

8) **[6 points] Speedup**

How many processors would you need to get 4x speedup on a program where 4/5 of the program is parallelizable? Is this possible?

You must show your work for any credit. For full credit give your answer as an integer (not a formula).

9) [14 points] In Java using the ForkJoin Framework, write code to solve the following problem:

• **Input**: An array of positive `ints`.
• **Output**: Print the two most commonly occurring **ones place digits**. If there is a tie in how common digits are, the smallest digit should be considered more common.

Input array: {2005, 5, 4, 35, 44, 3} most common: 5, second most common: 4
Input array: {7, 6, 17, 3} most common: 7, second most common: 3
Input array: {13, 6, 7, 6003, 17, 766} most common: 3, second most common: 6

- Do **not** employ a sequential cut-off: **the base case should process one element**. (You can assume the input array will contain at least two different `int`s.)
- Give a class definition, `TopTwoTask`, **along with any other code or classes needed**.
- Fill in the function `printTopTwo` **below.**

*****You may **NOT** use any **global data structures** or **synchronization primitives (locks).**
***Make sure your code has O(log n) span and O(n) work.**

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

class Main{
      public static final ForkJoinPool fjPool = new ForkJoinPool();

      // Prints the two most common digits occurring in the ones place.
      public static void printTopTwo (int[] input) {
         int top;
         int second;
         // Your code here:









         System.out.println("Most common ones place digit: " + top);
         System.out.println("Second most common ones place digit:" + second);
      }
}
```

Please fill in the function above and write your class(es) on the next page.

9) (Continued) Write your class(es) on this page.