Name: ____**Sample Solution**_____

Email address (UWNetID): _____

CSE 332 Winter 2018 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	Score
1	12	
2	9	
3	11	
4	10	
5	14	
6	13	
7	16	
8	9	
9	6	
Total	100	

1) [12 points total] Hash Tables

For a) and b) below, insert the following elements in this order: 14, 22, 33, 44, 13, 73, 28. For each table, TableSize = 10, and 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 values.

a) Separate chaining hash table – use a sorted linked list for each bucket where the values are ordered by **increasing value**

0	
1	
2	→ 22
3	→ 13 → 33 → 73
4	→ 14 → 44
5	
6	
7	
8	→ 28
9	

b) Quadratic probing hash table

0	
1	
2	22
3	33
4	14
5	44
6	
7	13
8	28
9	73

c) What is the load factor in Table b)? **0.7**

d) In a sentence or two, describe **double hashing**.

The first hash function determines the original location where we should try to place the item. If there is a collision, then the second hash function is used to determine the probing step distance as 1*h2(key), 2*h2(key), 3*h2(key) etc. away from the original location.

e) What is **re-hashing**? Would it be beneficial to implement **re-hashing** on a separate chaining hash table like the one shown above in part a)? Why or Why not?

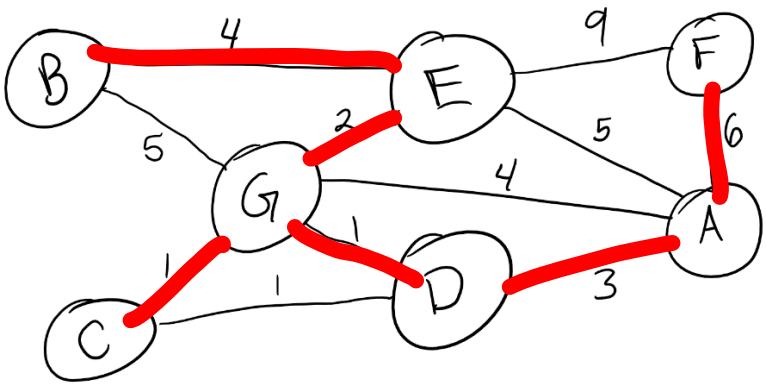
Re-hashing is creating a new table (usually ~twice as large and a prime number) and then rehashing the values from the original table and moding by the new tablesize and inserting into the new table. While a separate chaining hash table that uses linked lists as buckets will not ever fill up, the higher the load factor the longer the chains will get and performance will suffer. So yes, re-hashing would be beneficial.

f) What is one **disadvantage** quadratic probing has compared to linear probing?

In quadratic probing, if the table is more than half full (load factor = 0.5) then you are not guaranteed to be able to find a location to place the value in. In a linear probling hash table you can always find a location as long as the table is not completely full.

2) [9 points total] Graphs!

a) **[6 points]** Find a minimum spanning tree with Prim's algorithm **using vertex G as the starting node** (mark, circle, or highlight edges below to indicate they are in your minimum spanning tree). 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.*



	Cost	Prev	Known?
A	~~∞~~ ~~4~~ 3	~~G~~ D	~~F~~ T
B	~~∞~~ ~~5~~ 4	~~G~~ E	~~F~~ T
C	~~∞~~ 1	G	~~F~~ T
D	~~∞~~ 1	G	~~F~~ T
E	~~∞~~ 2	G	~~F~~ T
F	~~∞~~ ~~9~~ 6	~~E~~ A	~~F~~ T
G	~~∞~~ 0		~~F~~ T

b) **[1 point]** List the order the vertices are added to the known set:

G, C, D, E, A, B, F

c) **[1 point]** Pick a node you could start at to get a *different minimum spanning tree* than the one you found in part a). Which edge would be in this new tree that is ***not*** in your tree above? ***DO NOT DRAW THE WHOLE TREE.***

Starting node (for example, "Z"): C **Edge** (for example, "(X, Y)"): (C, D)

d) **[1 point]** Will Prim's starting at vertex G find a correct minimum spanning tree if the weight of edge (A,F) is set to be -6? (circle one)

YES **NO**

3) [11 points total] More Graphs!

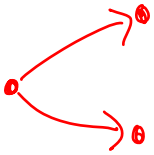
a) [2 points] If you needed to calculate the out-degree of all vertices in a graph, which representation would you prefer (circle one):

adjacency matrix or **adjacency list**

In a couple of sentences describe WHY?

In an adjacency matrix you would need to visit all V locations in the row for that vertex to calculate its out degree (so a total time of $O(|V|^2)$). In an adjacency list you would only need to visit the d (where d is the out-degree of a single node) nodes in the linked list linked for that vertex (so a total time of $O(|V| + |E|)$).

b) [2 points] Give an example of a directed graph with exactly two topological orderings and one node of in-degree zero.



c) [1 **point**] Let G be a connected, undirected, weighted graph. Convert G to a directed graph as follows: replace every undirected edge (u, v) with directed edges (u -> v) and (v -> u). The resulting graph is strongly connected.

TRUE FALSE

d) **[4 points]** What is the worst case running time of Dijkstra's algorithm described in lecture that:

i. Does NOT use a priority queue: $O(|V|^2 + |E|)$

ii. Uses a priority queue: $O(|V| \log |V| + |E| \log |V|)$

e) **[2 points]** Give an EXACT number (in terms of V) for:

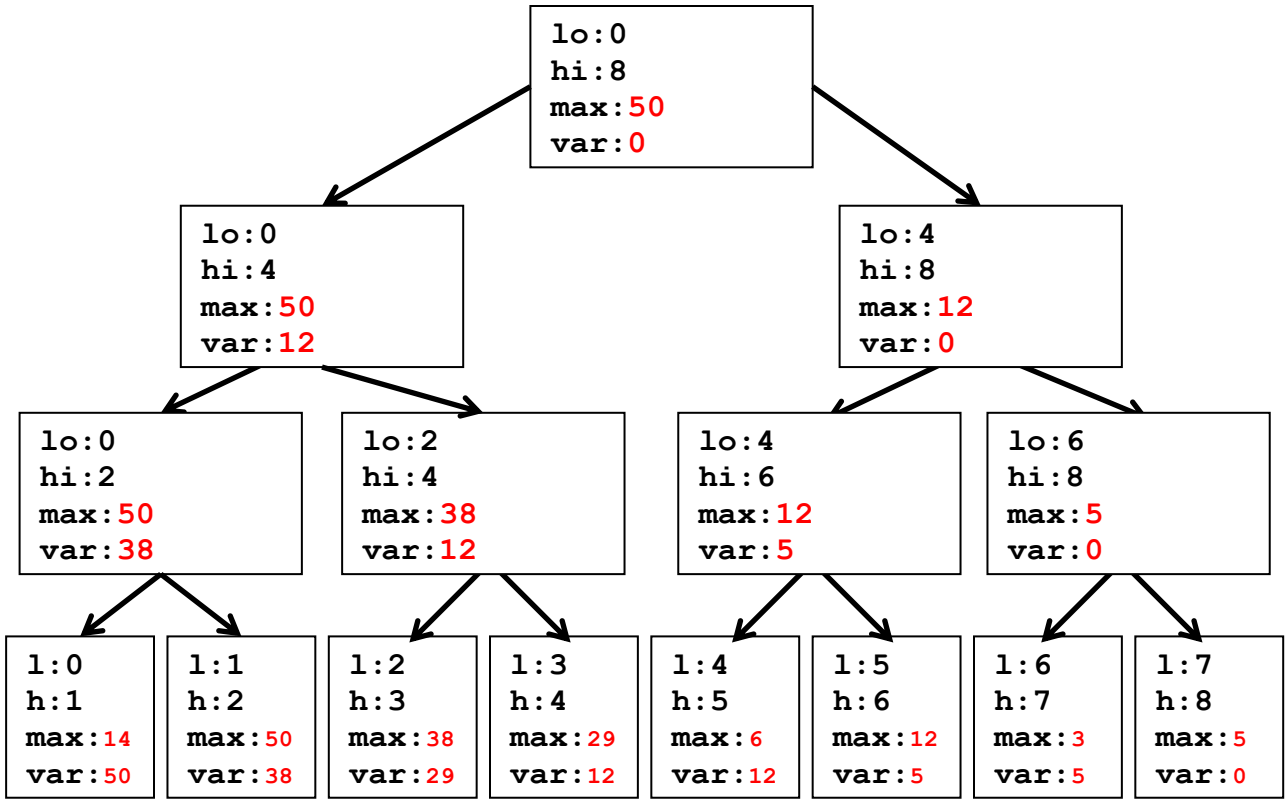
i. Maximum number of edges in an undirected graph without self-loops: $\frac{|V|(|V|+1)}{2} - |V|$

ii. Minimum number of edges in a weakly connected directed graph

$|V| - 1$

4) [10 points] Parallel "Suffix Max" (Like Prefix, but <u>from the Right</u> instead):

a) Given the following array as input, calculate the "suffix max" using an algorithm similar to the parallel prefix algorithm discussed in lecture. Fill the **output** array with the max of the values **contained in all of the cells to the right** (including the value contained in that cell) in the input array. The first pass of the algorithm is similar to the first pass of the parallel prefix code you have seen before. Fill in the values for **`max`** and **`var`** in the tree below. The output array has been filled in for you. Do not use a sequential cutoff. **You can assume that the array contains only positive integers.**



Index	0	1	2	3	4	5	6	7
Input	14	50	38	29	6	12	3	5
Output	50	50	38	29	12	12	5	5

b) How is the **`var`** value computed for the left and right children of a node in the tree. Give **<u>exact code</u>** (not just an English description) where **`p`** is a reference to the current tree node.

```
p.left.var =    Math.max(p.var, p.right.max)

p.right.var  =  p.var
```

c) How is **`output[i]`** computed? Give **<u>exact code</u>** assuming **`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.

```
output[i] = Math.max(leaves[i].max, leaves[i].var)
```

5) [14 points] In Java using the ForkJoin Framework, write code to solve the following problem:

• **Input**: An array of positive `ints`
• **Output**: an array of 10 `ints` containing a count of the **ones place digits** of the values in the Input array. The count of digit `i` will be in Output[`i`].
For example, if the input array is {2007, 13, 17, 24, 5, 17, 38, 407, 0, 7, 4, 17}, the output array (always containing exactly 10 `ints`) would be {1, 0, 0, 1, 2, 1, 0, 6, 1, 0}.

- 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, `CountOnesPlaceTask`, **along with any other code or classes needed**.
- Fill in the function `findOnesPlaces` **below.**

You may not use any global data structures or synchronization primitives (locks).

a) Write the code.
b) Answer this: Is this a **map** or a **reduction (circle one)**? **Why**?

We are reducing the problem from an array of size n down to an array of size 10.

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

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

      // Returns an array of 10 ints.  Where the ith element
      // contains a count of the number of times i appears in
      // the ones place in the values in input.
      public static int[] findOnesPlaces (int[] input) {

            return fjPool.invoke(new CountOnesPlaceTask(input, 0,
      input.length));

      }




}
```

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

Don't forget to answer b) on the previous page!

5) (Continued) Write your class on this page.

```
public static class CountOnesPlaceTask extends RecursiveTask<int[]>{
  int[] input;
  int lo;
  int hi;
  public CountOnesPlaceTask(int[] input, int lo, int hi) {
    this.input = input;
    this.lo = lo;
    this.hi = hi;
  }

  public int[] compute() {
    if (hi - lo <= 1) {
      int[] counts = new int[10];
      counts[input[lo] % 10] = 1;
      return counts;
    } else {
      int mid = lo + (hi - lo)/2;

      CountOnesPlaceTask left = new CountOnesPlaceTask(input, lo, mid);
      CountOnesPlaceTask right = new CountOnesPlaceTask(input, mid, hi);
      right.fork();
      int[] leftResult = left.compute();
      int[] rightResult = right.join();

      for (int i = 0; i < 10; i++) {
        leftResult[i] += rightResult[i];
      }
      return leftResult;
    }
  }

}
```

6) [13 points] Concurrency: The following class implements a Bank account class that keeps track of multiple Bank Accounts a user might have. Multiple threads could be accessing the same `BankAccounts` object.

```
public class BankAccounts {
      private Map<String, Double> acctsMap = new HashMap<>();
      ReentrantLock lock = new ReentrantLock();

      // Returns null if account does not exist
      public Double getBalance(String acctName) {
            lock.acquire();
            return acctsMap.get(acctName);
            Double balance = acctsMap.get(acctName);
            lock.release();
            return balance;
      }

      public Double withdraw(String acctName, Double amount) {
            lock.acquire();
            Double acctBalance = getBalance(acctName);

            if (acctBalance == null || acctBalance < amount) {
                  lock.release();
                  throw new InvalidTransactionException();

            }

            acctsMap.put(acctName, acctBalance - amount);
            lock.release();
            return amount;

      }

      // Deposit amount in acctName
      // Creates acctName if it does not already exist
      public void deposit(String acctName, Double amount) {
            lock.acquire();
            Double acctBalance = getBalance(acctName);

            if (acctBalance == null) {

                  acctBalance = 0.0;

            }

            acctsMap.put(acctName, acctBalance + amount);
            lock.release();

      }
}
```

6) (**Continued**)
a) Does the `BankAccounts` class above have (circle all that apply):

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

If there are any problems, describe them in 1-2 sentences.

There is a data race on the `acctsMap`, e.g. one thread could be reading it in `getBalance` while another thread could be writing it in `withdraw` or `deposit`. Similarly multiple threads could be writing it at the same time by calling `withdraw` or `deposit` simultaneously. A data race is a race condition.

b) Suppose we made the `withdraw` method **`synchronized`**, and changed nothing else in the code. Does this modified `BankAccounts` class above have (circle all that apply):

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

If there are any FIXED problems, describe why they are FIXED in 1-2 sentences. If there are any NEW problems, describe them in 1-2 sentences.

You can no longer have a race condition between two simultaneous calls to `withdraw`.

c) Modify the **code on the previous page** to use locks to avoid any of the potential problems listed above. Create locks as needed. Use any reasonable names for the locking methods you use. **DO NOT use `synchronized`**. You should create re-entrant lock objects as needed as follows (place this in your code as needed):

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

d) Clearly circle all of the **critical sections** in your **code on the previous page**.

7) [16 points] Sorting

a) **[2 points]** Give the ***recurrence*** for SEQUENTIAL Mergesort – worst case: (Note: We are NOT asking for the closed form.)

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$

b) **[3 points]** Give the ***recurrence*** for Quicksort (parallel sort & parallel partition) – worst case span: (Note: We are NOT asking for the closed form.)

$T(n) = T(n-1) + O(\log n)$

c) **[5 points]** Give the big-O runtimes requested below.

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

$O(n \log n)$ B) Heapsort – worst case

$O(n^2)$ C) Insertion Sort – worst case

$O(n)$ D) Bucket Sort – best case

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

d) **[1 point]** Is the version of Quicksort described in lecture a stable sort?

YES **NO**

e) **[2 points]** In 1-2 sentences, describe what it means for a sort to be stable?

In case of ties during the sort, the original ordering of values is preserved.

f) **[3 points] 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:

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

Answer: $O((\log_{radix} max\_value) \cdot (n + radix))$

8) **[9 points] P, NP, NP-Complete**

a) **[2 points]** "NP" stands for ____ **Nondeterministic Polynomial** _____

b) **[2 points]** 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) **[5 points]** For the following problems, circle **ALL** the sets they (most likely) belong to:

Finding the shortest path from one vertex to another vertex in a weighted directed graph	**NP**	**P**	NP-complete	None of these
Finding a cycle that visits each edge in a graph exactly once	**NP**	**P**	NP-complete	None of these
Determining if a program will run forever	NP	P	NP-complete	**None of these**
Finding the prefix sum of an array in parallel using 10 processors	**NP**	**P**	NP-complete	None of these
Finding a path that starts and ends at the same vertex that visits every vertex exactly once	**NP**	P	**NP-complete**	None of these

9) [6 points] Speedup

Your boss wants 111x speedup on a program of which 9/10 is parallelizable. What do you tell them? At least how many processors would you need?

Justify your answer with a computation. No credit given without an explanation.

$\frac{T_1}{T_\infty} = \frac{1}{S} = \frac{1}{1/10} = 10$ ← Maximum Speedup possible for a program with 1/10 that must be run sequentially.

Tell your boss that the max speedup possible on this program is 10x. So a 111x speedup will not be possible.