Name: _____

Email address (UWNetID): _____

CSE 332 Winter 2019 Kat's 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	11	17 an
2	10	16 an
3	9	17 an
4	10	18 wi
5	14	17 an
6	13	18 wi
7	6	18 wi
8	16	18 wi
9	11	
Total	100	

1) [11 points total] Hash Tables

For a) and b) below, insert the following elements in this order: 19, 48, 8, 27, 97, 7. 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**

b) Quadratic probing hash table

0	
1	
2	
3	
4	
5	
6	
7	→ 7 → 27 → 97
8	→ 8 → 48
9	→ 19

0	
1	97
2	8
3	
4	
5	
6	7
7	27
8	48
9	19

c) What is the load factor in Table a)? $6 / 10 = 0.6$

d) Would you implement lazy deletion on Table a)? In 1-2 sentences describe why or why not. Be specific.

No.

Cost of deletion is equivalent to find operation. Leaving nodes marked deleted just cause other operations to do extra traversal

e) In 1-2 sentences (or pseudo code) describe how you would implement re-hashing on Table b). Be specific.

We rehash when $\lambda > 0.5$ or when insertion fails.

Create new table with size ×2, rehash every value.

f) What is the big-O worst case runtime of an Insert in a hash table like Table a) containing N elements?

$O(N)$

g) What is the big-O worst case runtime of determining what the maximum value is in a hash table like Table b) containing N elements?

$O(N)$

2) [12 points total] Graphs!
   a) [2 points] What is the big-O running time of Dijkstra's algorithm (assuming an adjacency list representation) if:
   (i) A priority queue is used? $O(E \log V + V \log V)$

      (ii) An unsorted list is used? $O(EV + V^2)$

   b) [2 points] Which implementation of Dijkstra's (priority queue vs. unsorted list) is likely to be faster if the graph is known to be dense? **Explain your answer in ~one sentence for any credit**.

   Priority Queue is better.
   $O(\log V)$ is better than $O(V)$

   c) [2 points] Give a Minimum Spanning Tree (MST) of the graph below by highlighting the edges that would be part of the MST.
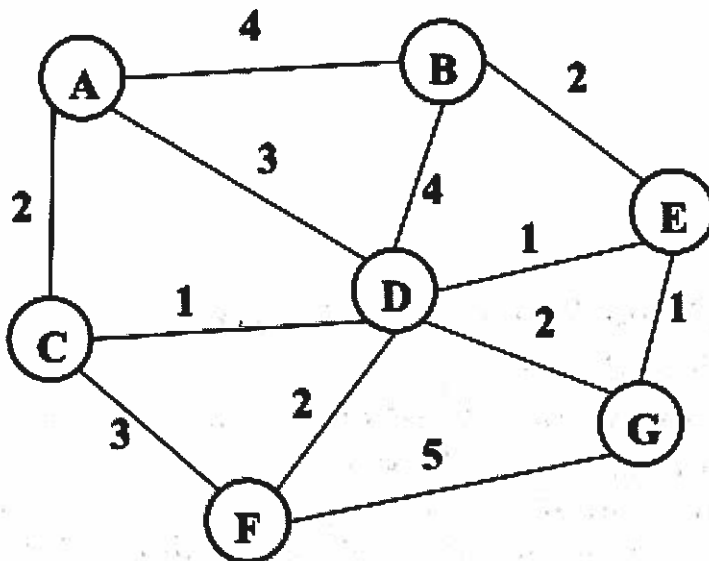
   d) [6 points] For (ii) and (iv) below, *you are given a perfect binary tree of height h containing n nodes*. Your answer should be an **exact formula**, such as 3/2 log h, or 5 × 3^n, not big-O notation.
   (i) Depth First Search: What is the name of the data structure used in DFS?
   stack
   (ii) What is the maximum size of that data structure during a DFS?
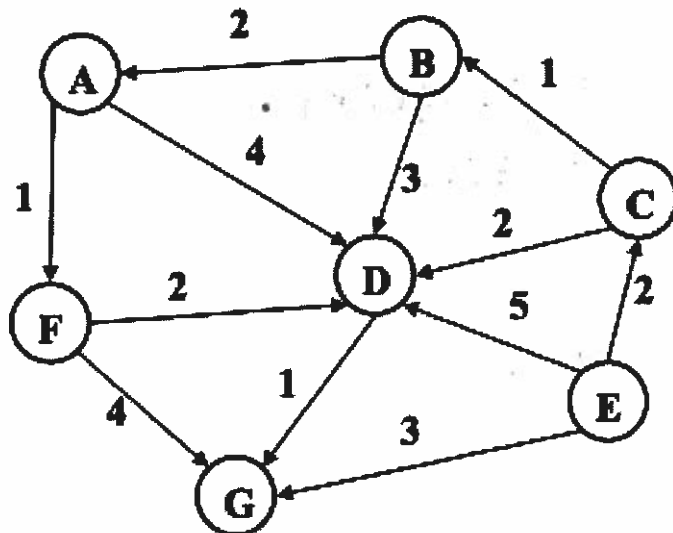 $h+1$

   (iii) Breadth First Search: What is the name of the data structure used in BFS?
   queue
   (iv) What is the maximum size of that data structure during a BFS?
   2^h

3) [9 points total] More Graphs! 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).

$$E\ C\ B\ A\ F\ D\ G$$

b) [3 points] In lecture we described an optimization to the topological sorting algorithm that used a queue. Your partner proposes that you use a priority queue instead of a FIFO queue. What would be the worst case running time of this new version of topological sort (assuming an adjacency list representation of the graph is used)? **For any credit, briefly describe your answer with pseudo code or in a couple of sentences.**

```
calculated Indegree Of All Vertices ();   // O(E+V)
build heap of vertices ();     // O(V)
for( ctr =0; ctr < numVertices; ctr++) {     // V times
      V= delete min ();       // O(log V)
      put v next in output      // O(1)
      for each w adjacent to v {   // E times total
            decreaseKey (w, 1);   // O(log V)
      }
}
```

c) [2 points] ASSUMING the edges above are undirected, give a valid <u>breadth first</u> search of this graph, starting at vertex A, using the algorithm described in lecture.
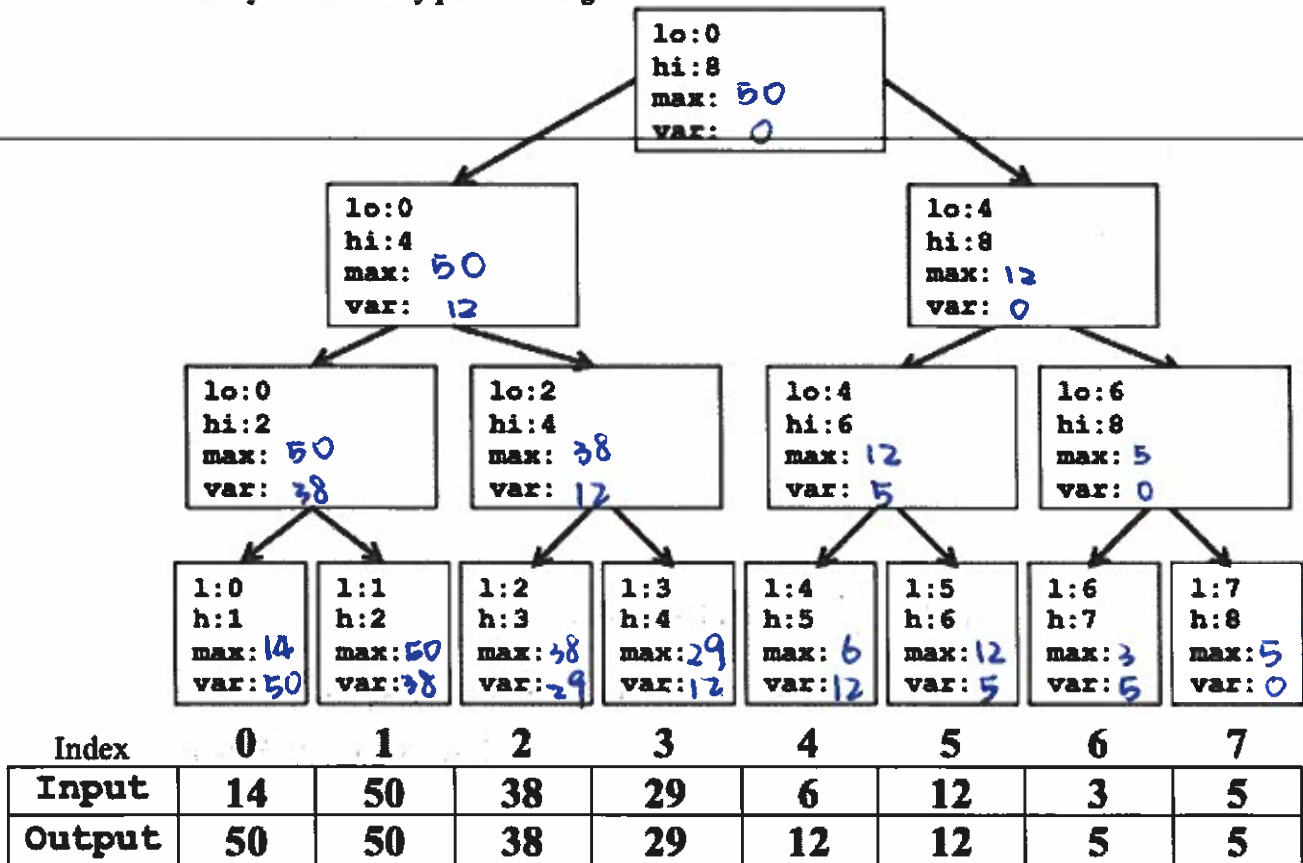
one possibility
A BDFCE G.

d) [2 points] ASSUMING the edges above are undirected, give a valid <u>depth first</u> search of this graph, starting at vertex A, using the non-recursive algorithm described in lecture.

One possibility

A B C E G F D

4) [10 points] Parallel "Suffix Max" (Like Prefix, but _from the Right_ 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.**

```
lo:0
hi:8
max: 50
var: 0
```

```
lo:0          lo:4
hi:4          hi:8
max: 50       max: 12
var: 12       var: 0
```

```
lo:0      lo:2      lo:4      lo:6
hi:2      hi:4      hi:6      hi:8
max: 50   max: 38   max: 12   max: 5
var: 38   var: 12   var: 5    var: 0
```

```
1:0     1:1     1:2     1:3     1:4     1:5     1:6     1:7
h:1     h:2     h:3     h:4     h:5     h:6     h:7     h:8
max:14  max:50  max:38  max:29  max:6   max:12  max:3   max:5
var:50  var:38  var:29  var:12  var:12  var:5   var:5   var:0
```

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 **exact code** (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 **exact code** 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 ints
> • **Output:** the array index of the rightmost number greater than zero in the Input array.

For example, if input array is {-3, 2, 5, -2, 7, 0, -4}, the output would be 4, since that is the index of the value 7. If there are no values greater than 0 in the array then -1 should be returned.

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

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

```java
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 the index of the rightmost positive value
    // in the array input. Returns -1 if no values > 0 in input.
    public static int findRightmostPos (int[] input) {
```

return fjPool.invoke(

new RightMostPosTask(input,

0, input.length));

```
    }

}
```

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

```java
public class RightmostPosTask extends RecursiveTask<Integer> {
    int[] array;
    int lo;
    int hi;
    public RightmostPosTask (int[] array, int lo, int hi) {
        this.lo = lo;
        this.hi = hi;
        this.array = array;
    }
    protected Integer compute() {
        if (hi-lo < 2) {
            if (array[lo] > 0) {
                return lo;
            } else {
                return -1;
            }
        } else {
            int mid = lo + (hi - lo)/2;
            RightmostPosTask left = new RmPT (array, lo, mid);
            RightmostPosTask right = new RmPT (array, mid, hi);
            left.fork();
            int rightRes = right.compute();
            int leftRes = left.join();

            return Math.max (rightRes, leftRes);
        }
    }
}
```

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 acutsMap.
one thread can read it in getBalance, another can
write it in withdraw or deposit.
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?

When there is a tie, 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:
$$O\left(\left(\log_{radix} max\_value\right) \cdot \left(n + radix\right)\right)$$

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 if 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