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.

17au

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	
8	
9	

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

c) What is the load factor in Table a)?

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

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

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

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?   _____

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?

(ii) An unsorted list is used?

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__.

c) **[2 points]** Give a Minimum Spanning Tree (MST) of the graph below by highlighting the edges that would be part of the MST.
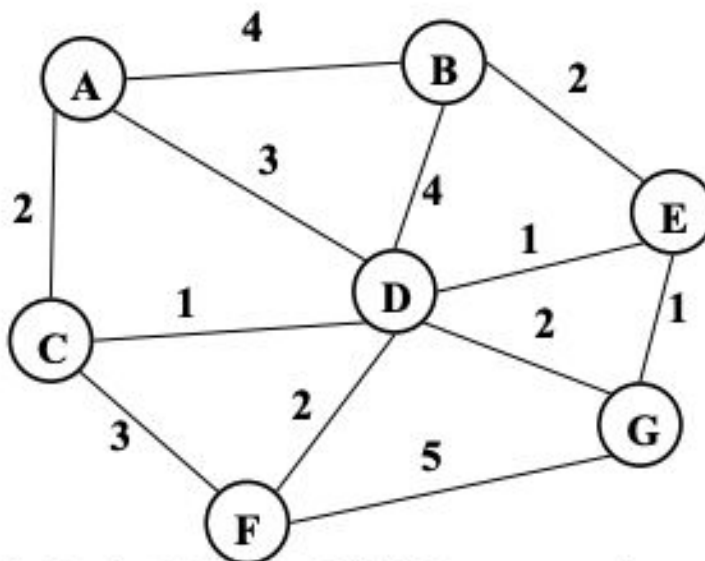
d) **[6 points]** For (ii) and (iv) below, *you are given a perfect binary tree of height h containing n nodes.* Your answer should be an **exact formula**, such as 3/2 log h, or 5 × 3^n, not big-O notation.

(i) Depth First Search: What is the name of the data structure used in DFS?

(ii) What is the maximum size of that data structure during a DFS ?

(iii) Breadth First Search: What is the name of the data structure used in BFS?

(iv) What is the maximum size of that data structure during a BFS ?

3) [9 points total] More Graphs!   Use the following graph for this problem:
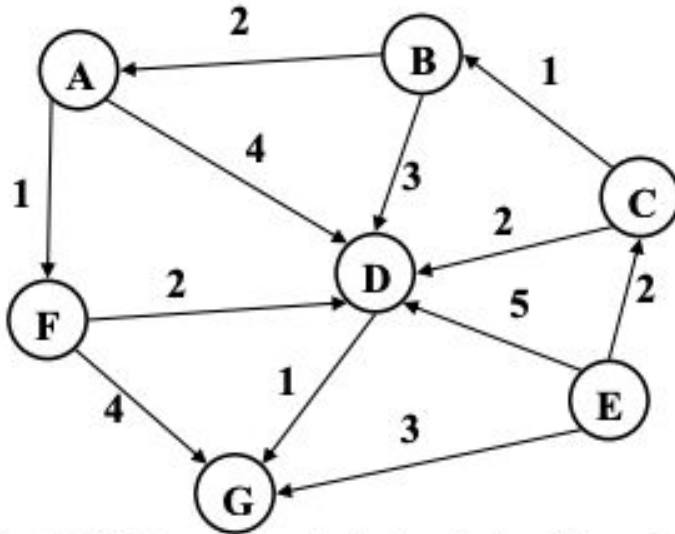
a) [2 points] List a valid **topological ordering** of the nodes in the graph above (if there are no valid orderings, state why not).
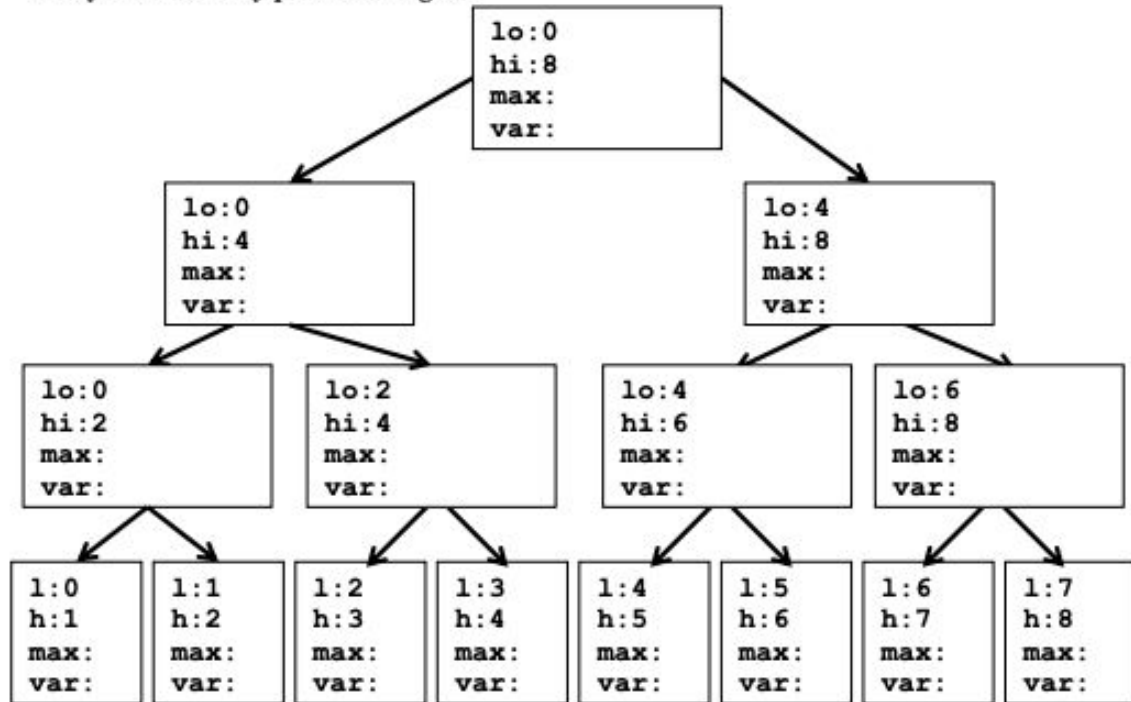
b) [3 points] In lecture we described an optimization to the topological sorting algorithm that used a queue. Your partner proposes that you use a priority queue instead of a FIFO queue. What would be the worst case running time of this new version of topological sort (assuming an adjacency list representation of the graph is used)? **For any credit, briefly describe your answer with pseudo code or in a couple of sentences.**

c) [2 points] ASSUMING the edges above are undirected, give a valid breadth first search of this graph, starting at vertex A, **using the algorithm described in lecture**.

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

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:
                          var:


            lo:0                              lo:4
            hi:4                              hi:8
            max:                              max:
            var:                              var:


    lo:0          lo:2              lo:4              lo:6
    hi:2          hi:4              hi:6              hi:8
    max:          max:              max:              max:
    var:          var:              var:              var:


 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:   max:   max:   max:     max:   max:   max:   max:
 var:   var:   var:   var:     var:   var:   var:   var:
```

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 =

p.right.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] =

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) {
```

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

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<>();


      // Returns null if account does not exist
      public Double getBalance(String acctName) {

            return acctsMap.get(acctName);



      }

      public Double withdraw(String acctName, Double amount) {

            Double acctBalance = getBalance(acctName);

            if (acctBalance == null || acctBalance < amount) {

                  throw new InvalidTransactionException();

            }

            acctsMap.put(acctName, acctBalance - amount);

            return amount;

      }

      // Deposit amount in acctName
      // Creates acctName if it does not already exist
      public void deposit(String acctName, Double amount) {

            Double acctBalance = getBalance(acctName);

            if (acctBalance == null) {

                  acctBalance = 0.0;

            }

            acctsMap.put(acctName, acctBalance + amount);

      }
}
```

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.

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.

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.)

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

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

_____ A) Quicksort (parallel sort & parallel partition) − best case span

_____ B) Heapsort − worst case

_____ C) Insertion Sort − worst case

_____ D) Bucket Sort − best case

_____ 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?

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:

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

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

b) **[2 points]** What does it mean for a problem to be in NP?

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