CSE 332 Spring 2024: Final Exam

Name: _____

UWNetID: _____

Instructions:

- The allotted time is 1 hour and 50 minutes.
- Please do not turn the page until the staff says to do so.
- This is a closed-book and closed-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- When provided, write your answers in the box or on the line provided.
- Unless otherwise noted, every time we ask for an O, Ω, or Θ bound, it must be simplified and tight.
- Unless noted, assume all data structures are implemented as described in lecture.
- For answers that involve bubbling in a ◯, fill in the shape **completely**.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan the exams; edges often get cropped.
- A formula sheet has been included at the end of the exam.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Question #	Topic	Points	Page #
Q1	**Short Answer**	**10**	**1-2**
Q2	**Hashing**	**10**	**3**
Q3	**Sorting**	**9**	**4**
Q4	**Undirected graphs**	**11**	**5-6**
Q5	**Directed graphs**	**14**	**7-8**
Q6	**Fork-Join**	**14**	**9-11**
Q7	**Parallel Prefix**	**7**	**12-13**
Q8	**Concurrency**	**14**	**14-16**
Q9	**P/NP**	**5**	**17**
Extra Credit		**(1)**	**18**
Total		**94**	

Useful Math Identities

Summations

1. $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ for $|x| < 1$
2. $\sum_{i=0}^{n-1} 1 = \sum_{i=1}^{n} 1 = n$
3. $\sum_{i=0}^{n} i = 0 + \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
4. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
5. $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
6. $\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$
7. $\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$

Logs

A few useful formulas

1. $x^{\log_x n} = n$
2. $a^{\log_b c} = c^{\log_b a}$
3. $\log_b a = \frac{\log_d a}{\log_d b}$

Q1: Short Answer (10 points)

We will only grade what is in the provided answer box.

a) Give a simplified, tight big-O bound for $f(N) = N\,log_2(2^N) + (log\,N)^2$

O( ) **)**

b) What is the minimum number of nodes in a binary search tree of height ***h***?

c) In a B-Tree with M=1000, L=15, and a height of 4, how many **disk blocks** will we access as part of a `find` operation, assuming we size B-Tree nodes such that they each fit nicely onto one disk block?

d) Alice is a nurse in the emergency department of the hospital; she needs to triage patients based on the severity of their emergencies. What **abstract data type** that we learned about this quarter should Alice use for her application?

e) What is the **work** of parallel partitioning ***n*** elements into three groups during **parallel** quicksort: elements less than pivot, elements greater than pivot, and the pivot?

O( ) **)**

f) If 80% of your algorithm can be parallelized, but 20% cannot be parallelized, what is the **maximum** speedup that you can expect if you use more and more processors?

Q1 (continued)

g) What is the **worst-case** complexity of **inserting** an element into a dictionary with ***n*** elements that is implemented with a chaining hash table, assuming that the hash table chains are linked lists?

O( )

h) What is the **span** of the parallel pack algorithm?

O( )

i) What is the **worst-case** complexity of doing a radix sort on ***n*** elements with a radix of ***R*** and a maximum element length of ***P***?

O( )

j) Give the big-O complexity for the following function in terms of **N**, the size of `arr1`, and **M**, the size of `arr2`.

```
public List<String> magic(int[] arr1, String[] arr2) {
    List<String> result = new ArrayList<>();
    for (int v : arr1) {
        for (int z : arr1) {
            if (z > v) {
                result.add(z + "!");
            }
        }
        if (v < 10) {
            for (int w : arr2) {
                if (w.length() > v) {
                    result.add(w);
                }
            }
        }
    }
    return result;
}
```

O( )

Q2: Hashing (10 points)

a) (6 points) Insert **9, 5, 20, 16, 2, 22, 12** into each of the hash tables below (one quadratic probing, one separate chaining using a linked list that adds at the end). For each table, TableSize = **9**, and you should use the primary hash function **h(k) = k%9**. If an item cannot be inserted into the table, please indicate this in the appropriate box and continue inserting the remaining values.

Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	

Separate Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	

Values that could not be inserted:

b) (1 point) What is the load factor for the **probing** hash table in part a)? You may leave your answer unsimplified.

c) (1 point) What is the load factor for the **chaining** hash table in part a)? You may leave your answer unsimplified.

d) (2 points) Based on the guidelines we discussed in lecture, should we resize either hash table on the next insertion?

◯ probing ◯ chaining ◯ both ◯ neither

Q3: Sorting (9 points)

Given the array below, perform one quicksort partition by answering the following questions:

28	91	58	60	68	73	13	82	50	72	97

a) (1 point) Using the median-of-3 strategy of selecting the pivot, what **value** will we select as the pivot at the first partition step?

b) (3 points) In the array below, show the result of partitioning based on the pivot you selected in part (a). You do NOT need to recurse; **only do the first partition step**.

c) (1 point) Which of the following apply to quicksort as taught in lecture? Select all that apply.

◯ in-place sort ◯ stable sort
◯ comparison sort ◯ none of these

d) (1 point) Give the recurrence for sequential quicksort in the **worst** case, in terms of n. You do not need to include constants.

For each of the scenarios described below, indicate which type of sort would be the **most appropriate**, i.e. *fastest*.

e) (1 point) Sorting hotels on a travel website based on their star rating.

◯ insertion sort ◯ merge sort ◯ quicksort ◯ bucket sort

f) (1 point) Sorting a mostly-already-sorted list of UW students by name.

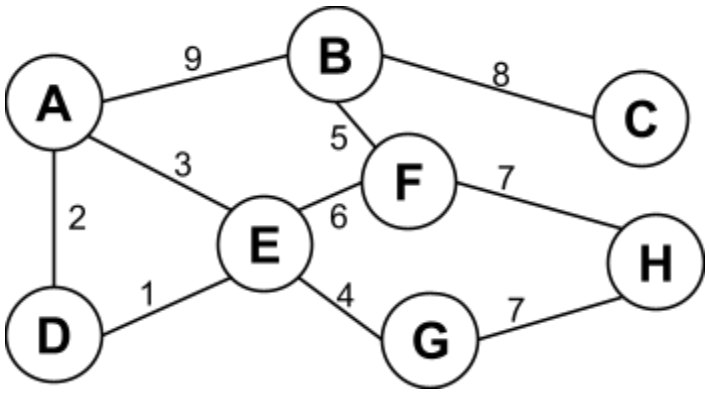
◯ insertion sort ◯ merge sort ◯ quicksort ◯ bucket sort

g) (1 point) Sorting a list of movies by title while also maintaining the current relative ordering.

◯ insertion sort ◯ merge sort ◯ quicksort ◯ bucket sort

Q4: Undirected Graphs (11 points)

Use the following graph for the problems on this page:



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) If we run Kruskal's algorithm on the tree above, what are the **weights** of the first three edges added to the minimum spanning tree?

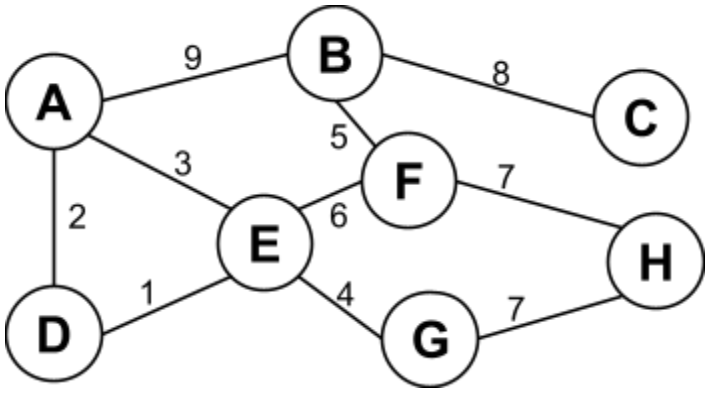
First: Second: Third:

c) (1 point) Give the **total cost** of a minimum spanning spanning tree in the graph above.

d) (1 point) Prim's algorithm can sometimes produce a different minimum spanning tree from Kruskal's algorithm. Give **one** edge whose weight we could change in the above tree so that Kruskal's algorithm and Prim's always give the same result. Indicate the edge as (X, Y).

Q4 (continued)

This is the same graph as on the previous page, for reference:

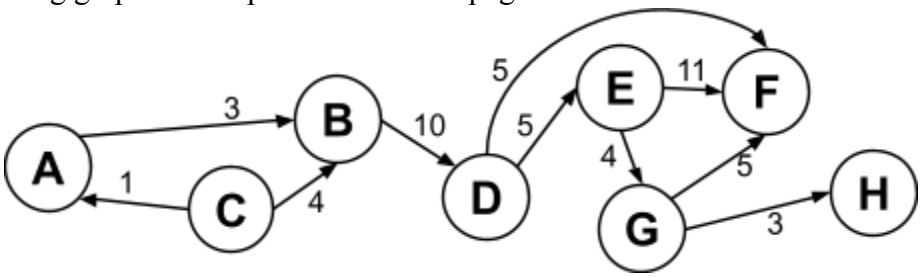


e) (2 points) ASSUMING the edges above are **unweighted**, give a valid **breadth first search** of this graph, **starting at vertex E**, using the algorithm described in lecture. **When adding elements to the data structure, you should break ties by choosing the lexicographically smallest letter first**; ex. if A and B were tied, you would add A to the data structure first. You only need to show the final breadth first search, in the format of [W, X, Y, Z, etc].

f) (2 points) ASSUMING the edges above are **unweighted**, give a valid **depth first search** of this graph, **starting at vertex E**, using the algorithm described in lecture. **When adding elements to the data structure, you should break ties by choosing the lexicographically smallest letter first**; ex. if A and B were tied, you would add A to the data structure first. You only need to show the final depth first search, in the format of [W, X, Y, Z, etc].

Q5: Directed Graphs (14 points)

Use the following graph for the problems on this page:



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) (1 point) Which node has the greatest **in-degree**?

c) (2 points) Would an adjacency list or an adjacency matrix representation be more appropriate for this graph from a space usage perspective? Please explain why in 1 sentence.

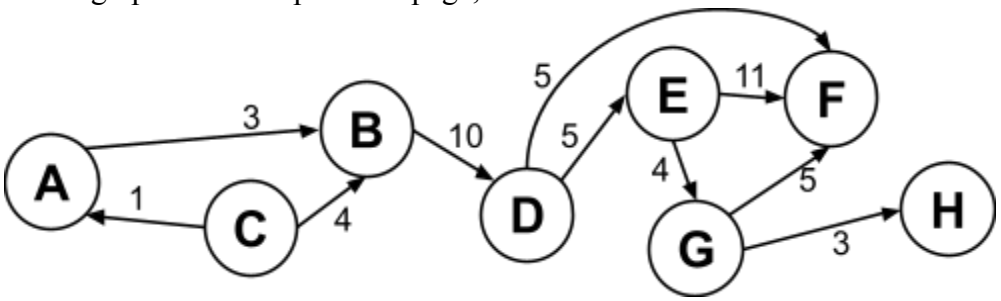
◯ adjacency list ◯ adjacency matrix

d) (2 points) Give an adjacency list representation of the graph above.

A
B
C
D
E
F
G
H

Q5 (continued)

This is the same graph as on the previous page, for reference:



e) (5 points) Step through Dijkstra's Algorithm to calculate the **single source shortest path from A** to every other vertex. Break ties by choosing the lexicographically smallest 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*. Include all intermediate states, but make sure the **final** distance and predecessor are clear in the table below.

Vertex	Known	Distance	Predecessor
A			
B			
C			
D			
E			
F			
G			
H			

f) (1 point) In what order would Dijkstra's algorithm mark each node as known?

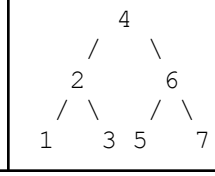
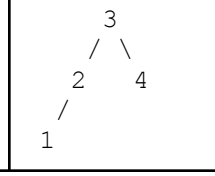
g) (1 point) List the shortest path from A to F (Give the actual path NOT the cost.)

Q6: ForkJoin (14 points)

In Java using the ForkJoin Framework, write code to solve the following problem:

- Input: A sorted array of unique integers.
- Output: The root of an AVL Tree containing all of the integers.

Examples:

Input	`[1, 2, 3, 4, 5, 6, 7]`	`[1, 2, 3, 4]`	`[1]`	`[]`
Output			1	null

Notes:

- **Do NOT use a sequential cutoff/method.** Your recursive class should divide & conquer all the way to the bottom and **the base case should process no more than one element.**
- Make sure your code has $O(log\ n)$ span and $O(n)$ work, where n is the number of elements in the array.
- You may NOT use any global data structures or synchronization primitives (locks).

(10 points) Fill in the underline in the function `buildAVLFromSortedArray` below, and add your class on the next page:

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

public class Main {
  public static final ForkJoinPool fjPool = new ForkJoinPool();
  public static class AVLNode { // You should use this class
    int key;
    int height;
    AVLNode left, right;

    // constructor
    AVLNode(int k, int h, AVLNode l, AVLNode r) { ... }
  }

  public static AVLNode buildAVLFromSortedArray(int[] arr) {

    return fjPool.invoke(new Task(_____________________________));
  }

  // Your class goes here (write it on the next page)
}
```

Q6 (continued)

Write your class here:

```
public static class Task extends _____ {

  private int[] arr;
  private int lo;
  private int hi;


  public Task(int[] arr, int lo, int hi) {
    this.arr = arr;
    this.lo = lo;
    this.hi = hi;
  }

  public _____ compute() {









  }
}
```

Q6 (continued)

(1 point) Give the recurrence relation $T(n)$ for your ForkJoin code in terms of n, the number of elements in the array, assuming perfect parallelism. You do not have to count operations; use *c* for constants.

$T(n) =$

(2 point) In the preceding ForkJoin code, we told you not to use a sequential cutoff. In real situations, we always want to use a sequential cutoff. In no more than 1-2 sentences, explain why, in general, it is better to use a sequential cutoff.

(1 point) Now we want to use a sequential cutoff for the reasons you mentioned in your answer to the previous question. It turns out that a sequential, recursive implementation would look *very* similar to the code that you already wrote. We don't want to duplicate code, though. Describe how you could modify your code to employ a sequential cutoff **WITHOUT duplicating code or adding a `sequential` method**. You may either describe the change in a sentence or two or write the code for the update.

Q7: Parallel Prefix (7 points)

Given the following array of strings as input, perform a parallel prefix algorithm to fill in the output array with the **total length of all the even-length strings in all of the cells to the left** (including the value contained in that cell) in the input array. Do not use a sequential cutoff. For example, for input:

```
{"hi", "my", "name", "seems", "to", "be", "Cookie", "Monster"}
```

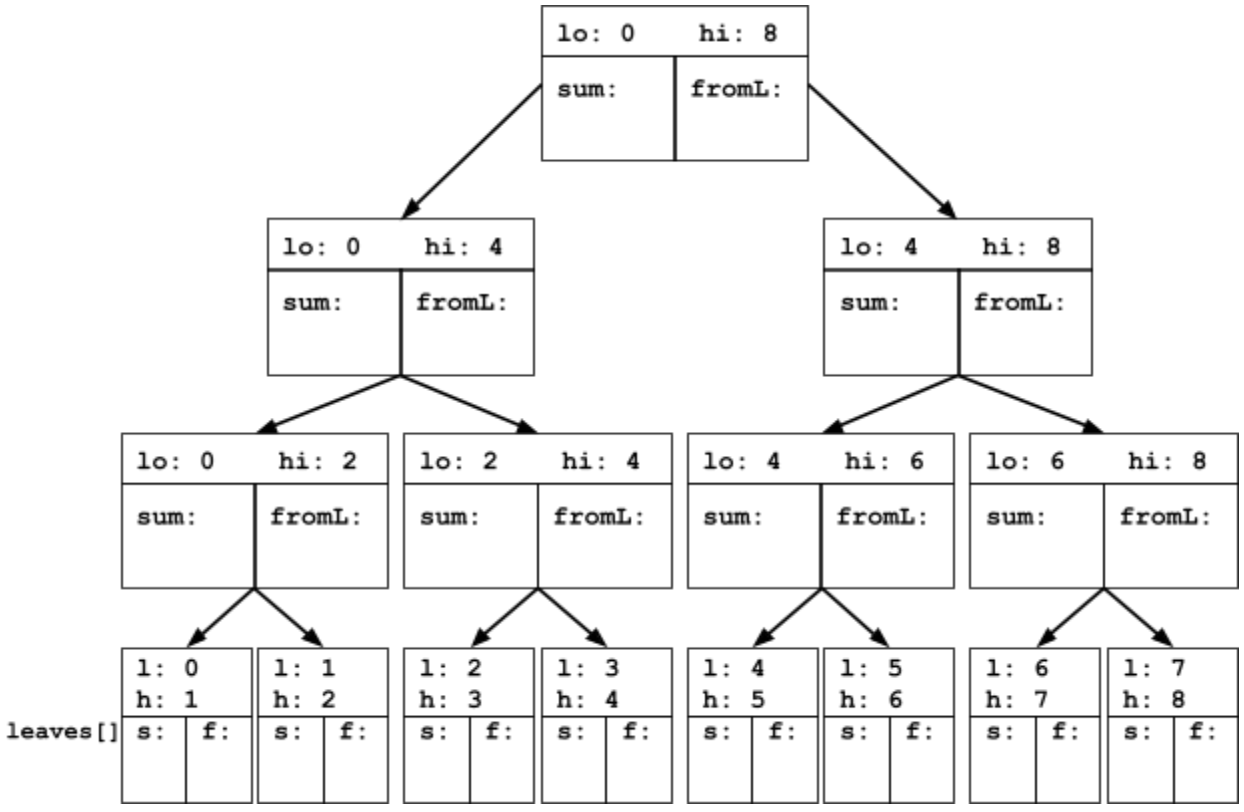
the output should be the following:

```
{2, 4, 8, 8, 10, 12, 18, 18}
```

Notice how we **skip** the odd-length strings "seems" and "Monster" - they do not contribute to the overall cumulative sum.

Note: This question continues on the next page.

a) (3 points) Fill in the values for `sum`, `fromL` (aka "fromLeft"), and the output array in the picture below. Note that problems (b)-(e), on the next page, ask you to give the formulas you used in your calculation.



Index	0	1	2	3	4	5	6	7
input	My	fav	lecture	this	quarter	is	parallel	prefix
output								

Q7 (continued)

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 on the previous page.

b) (1 point) Give pseudocode for how you assigned a value to **p[i].sum**.

c) (1 point) Give code for assigning **p.left.fromL**.

p.left.fromL =

d) (1 point) Give code for assigning **p.right.fromL**.

p.right.fromL =

e) (1 point) 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] =

Q8: Concurrency (14 points)

The `TicketManager` class below is used for selling tickets to events. Multiple threads (customers) might be accessing the same `TicketManager` object in order to buy tickets.

```
1   public class TicketManager {
2     private int[] seatsReserved;
3     private ReentrantLock[] seatLocks;
4     private int nextReservationNumber;
5
6     public TicketManager(int numSeats) {
7       seatsReserved = new int[numSeats];
8       seatLocks = new ReentrantLock[numSeats];
9       for (int i = 0; i < numSeats; i++) {
10        seatLocks[i] = new ReentrantLock();
11      }
12      nextReservationNumber = 1;
13    }
14    public synchronized boolean seatIsAvailable(int seatNum) {
15
16      return seatsReserved[seatNum] == 0;
17
18    }
19
20    private synchronized void markUnavailable(int seatNum,
21                                              int confirmationNumber) {
22
23      seatsReserved[seatNum] = confirmationNumber;
24
25    }
26
27    private synchronized int getConfirmationNumber() {
28
29      return nextReservationNumber++;
30
31    }
32
33    public int reserve(int seatNum) {
34
35      if (seatIsAvailable(seatNum)) {
36
37        int confirmationNumber = getConfirmationNumber();
38
39        markUnavailable(seatNum, confirmationNumber);
40
41        return confirmationNumber;
42
43      }
44
45      return -1;
46
47    }
48  }
```

Q8 (continued)

a) (4 points) Does the `TicketManager` class above have (bubble in **all** that apply):

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

If there are any problems, describe what can go wrong for **each** problem selected in 1-2 sentences. You **must** use line numbers where appropriate.

You decide to add another method to the `TicketManager` class to buy more than one ticket in a single order, so that we can support group reservations.

```
50  public int reserveGroup(int[] seatNums) {
51      for (int i = 0; i < seatNums.length; i++) {
52          seatLocks[seatNums[i]].lock();
53          if (!seatIsAvailable(seatNums[i])) {
54              for (int j = i; j >= 0; j--)
55                  seatLocks[seatNums[j]].unlock();
56              return -1;
57          }
58      }
59      int confirmationNumber = nextReservationNumber++;
60      for (int seat : seatNums) {
61          markSeatUnavailable(seat, confirmationNumber);
62          seatLocks[seat].unlock();
63      }
64      return confirmationNumber;
65  }
```

Answer the question about this method on the following page.

Q8 (continued)

b) (4 points) Does adding the `reserveGroup` method cause a **new** (bubble in **all** that apply):

◯ race condition ◯ potential for deadlock
◯ data race ◯ none of these

If there are any problems, describe what can go wrong for **each new problem** selected in 1-2 sentences. You **must** use line numbers where appropriate.

c) (6 points) Modify the **code above in part b) and on the first page** 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**.

- You should use `synchronized` or the provided `seatLocks` appropriately to provide locking.
- Draw arrows if needed.
- Cross things out if needed.
- Your code does not need to be perfect Java; **pseudocode for anything that does not have to do with concurrency is acceptable**.

Q9: P and NP (5 points)

a) (1 point) "NP" stands for

b) (1 point) Draw a diagram demonstrating how (we are pretty sure) the sets P, NP, and NP-Complete, overlap / don't overlap with each other.

For the following two problems, bubble in **ALL** the sets each problem belongs to.

c) (1 point) Find a path through a graph that visits each vertex exactly once, and starts and ends at the same vertex.

◯ P ◯ NP ◯ NP-complete ◯ none of these

d) (1 point) Find a path through a graph that visits each edge exactly once, and starts and ends at the same vertex.

◯ P ◯ NP ◯ NP-complete ◯ none of these

The following question asks you to consider the outcome of a hypothetical research project.

e) (1 point) You do some deep thinking and discover that the Bellman Ford algorithm to find a shortest path in a graph with negative-weight edges is in NP! We can conclude from this that:

◯ P = NP ◯ P ≠ NP ◯ Neither

Extra Credit (1 point)

In the space below, give a representation (drawing, story, poem, etc) of what you want to be doing or will be doing on the first day of summer break.