CSE 332 Summer 2025 Exam 2

Name: _____

UW NetID: _____(@uw.edu)

Instructions:

- The allotted time is 1 hour.
- 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.
- For answers that involve bubbling in a ◯ or ☐, make sure to 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 your exams and edges often get cropped off.
- A formula sheet has been included at the end of the exam.

Advice:

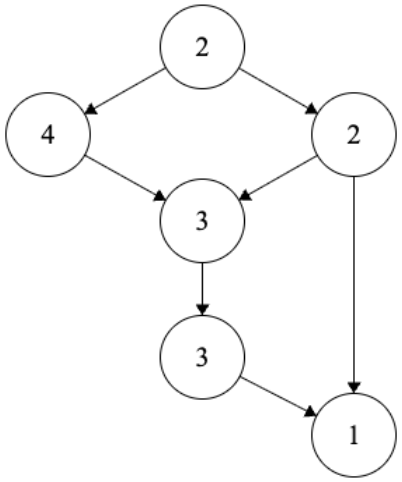
- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Q1: Short-answer questions

- For questions asking you about runtime, give a simplified, tight Big-O bound. For example, $O(5n^2 + 7n + 3)$ (not simplified) or $O(2^{n!})$ (not tight enough) are unlikely to get points. Unless otherwise specified, all logs are base 2.

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

a. Give the span of this task graph.



b. Give the **maximum** number of edges in an **undirected, connected, acyclic** graph with N vertices.

c. If the span of a parallel algorithm is **Ω**(n), then the work is **Ω**(n).

○ True ○ False

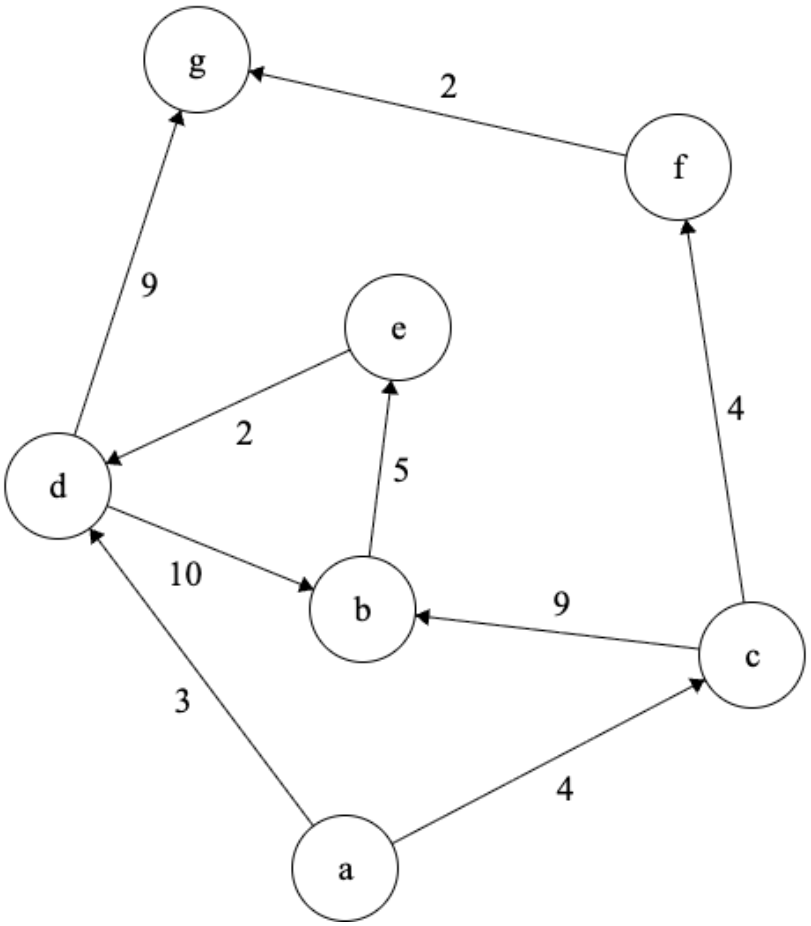
d. In Java, if a variable **`x`** references a **`String`**, then there can be no data races on that **`String`**.

○ True ○ False

e. What fraction of a program must be parallelizable in order to get **4x** speedup on **8** processors? Give your answer as a fraction.

Q2: Graphs

Use the following graph for the problems on this page:



a) 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*.** Make sure the final distance and predecessor are clear in the table below.

Vertex	Known	Distance	Predecessor
a			
b			
c			
d			
e			
f			
g			

Q2 continued:

b) In what order would Dijkstra's algorithm mark each node as *known***?**

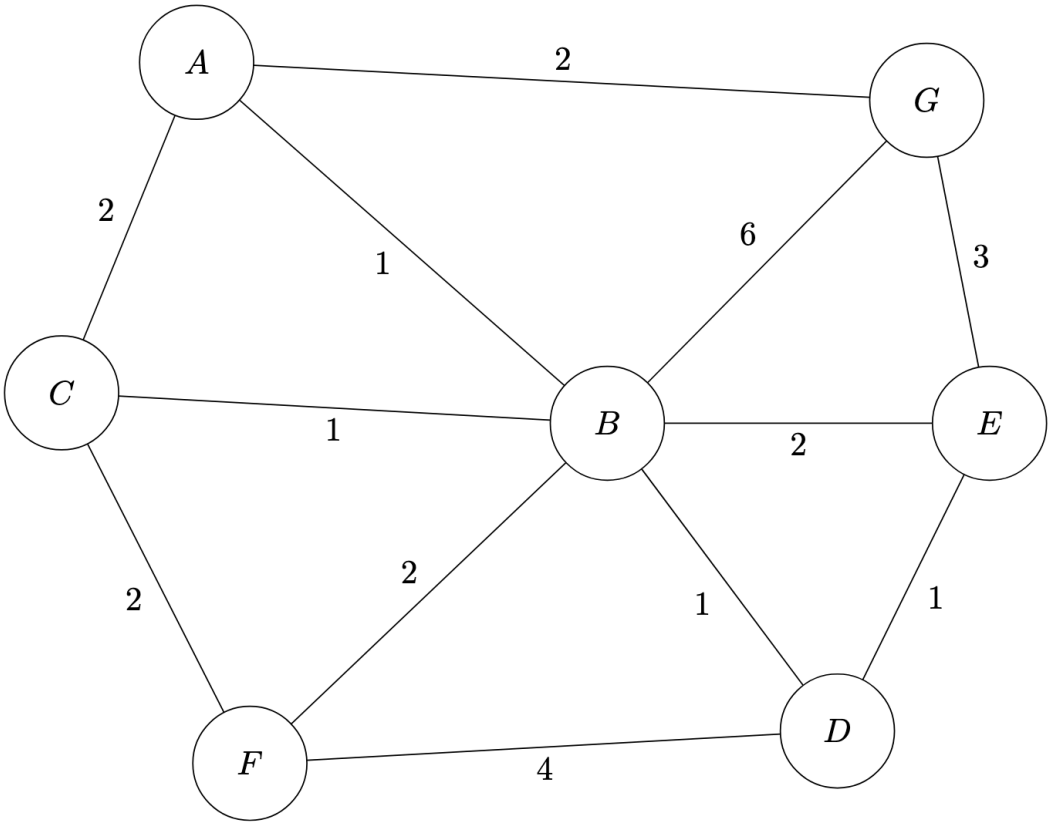
_____, _____, _____, _____, _____, _____, _____

c) List the **shortest path** (**NOT** its cost) from **a** to **e** given by Dijkstra's algorithm.

d) Is this graph **strongly connected**? **Explain your answer in 1-2 sentences for any credit.**

◯ Yes ◯ No

Q2 continued: Use the following graph for the problems on this page:



e) If we run Kruskal's algorithm, which of the following edges could be **<u>the last edge added</u>** to the Minimum Spanning Tree? **Bubble all that apply.** Assume any ties are broken randomly.

☐ AB ☐ AC ☐ AG ☐ BC ☐ BF ☐ BD
☐ BE ☐ BG ☐ CF ☐ DF ☐ DE ☐ EG

f) What is the **total cost** of a minimum spanning tree in the graph above? ☐

g) Treating the edges as **unweighted**, perform a **breadth first search** of this graph, **<u>starting at vertex F</u>**, using the algorithm described in lecture. Give the order in which nodes are **removed from the data structure**. When adding elements to the data structure, you should **break ties by choosing the lexicographically smallest letter first** (e.g. if A and B were tied, you would add A first). You only need to show the final ordering.

_____, _____, _____, _____, _____, _____, _____

Q3: ForkJoin

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

- **Input**: A non-empty array of non-empty lowercase Strings.
- **Output**: Returns the index of the first String that starts and ends with the same character, or -1 if no such String exists.

For example, Input array: {"foo", "bar", "dad", "a", "mom"} returns 2 and Input array: {"i", "am", "the", "best"} returns 0. Java's **`charAt(int i)`** and **`length()`** methods for Strings return the i-th character in a String and the length of a String, respectively.

- **Do **not** employ a sequential cut-off: **the base case should process exactly one element**.**
  - i.e. you do not **need** to employ a sequential method and you can assume that the code will never process more than one element
- Give a class definition, **`SameTask`**, along with any other code or classes needed.
- Fill in the _____ in the function **`findSame`** **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;

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

    // Returns the index of the first String that starts and ends
    // with the same character, or -1 if no such String exists.
    public static int findSame(String[] input) {
       return fjPool.invoke(new SameTask(_____));
    }
}
```

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

****Don't forget to fill in the blank line above!!!!**

Write your class here:

```
public class SameTask extends  ____________________________  {
    // Fields go here




    public SameTask(_____________________________________________) {




    }

    public _____________compute() {
```

Q4: Concurrency

The following class helps the delivery team manage packages. **Assume `ArrayList` is thread-safe.**

```
 1  public class Delivery {
 2      private final List<Package> packages;
 3      private final int capacity = 10;
 4      private boolean isEmpty;
 5      private final Object capacityLock = new Object();

 6      public Delivery() {
 7          packages = new ArrayList<>();
 8          isEmpty = true;
 9      }

10      public void addPackage(Package p) {
11          synchronized (capacityLock) {
12              if (packages.size() >= capacity) {
13                  throw new IllegalStateException("capacity reached");
14              }
15          }
16          packages.add(p);
17          isEmpty = false;
18      }

19      public Package deliverNext() {
20          if (isEmpty) {
21              throw new IllegalStateException("No packages to deliver");
22          }

23          Package next = packages.remove(0);
24          if (packages.size() == 0) {
25              isEmpty = true;
26          }
27          return next;
28      }

29      public int numPackages() {
30          return packages.size();
31      }
32  }

33  class Package {
34      private String name;

35      public Package(String name) {
36          this.name = name;
37      }
38  }
```

Q4 continued:

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

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

b) Does the **`Delivery`** class above have (bubble in all that apply):

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

Give an explanation for each box you checked for part (b) (1-2 sentences each). Refer to line numbers in your explanation. Be specific!

c) Jacklyn decided to eliminate the potential concurrency issue. This is her new **`Delivery`** class:

```
public class Delivery {
    // …FIELDS AND CONSTRUCTOR OMITTED…
    public void addPackage(Package p) {
        synchronized (capacityLock) {
            synchronized (packages) {
                if (packages.size() >= capacity) {
                    throw new IllegalStateException("...");
                }
            }
        }
        synchronized (packages) {
           packages.add(p);
           isEmpty = false;
        }
    }
    public Package deliverNext() {
        synchronized (packages) {
           synchronized (capacityLock) {
                  if (isEmpty) {throw new IllegalStateException("..."); }
            }
            Package next = packages.remove(0);
            if (packages.size() == 0) { isEmpty = true; }
           return next;
        }
    }
    public int numPackages() {
        synchronized (packages) { return packages.size(); }
    }
}
```

Does the **`Delivery`** class above have (bubble in all that apply):

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

Q5: Parallel Prefix

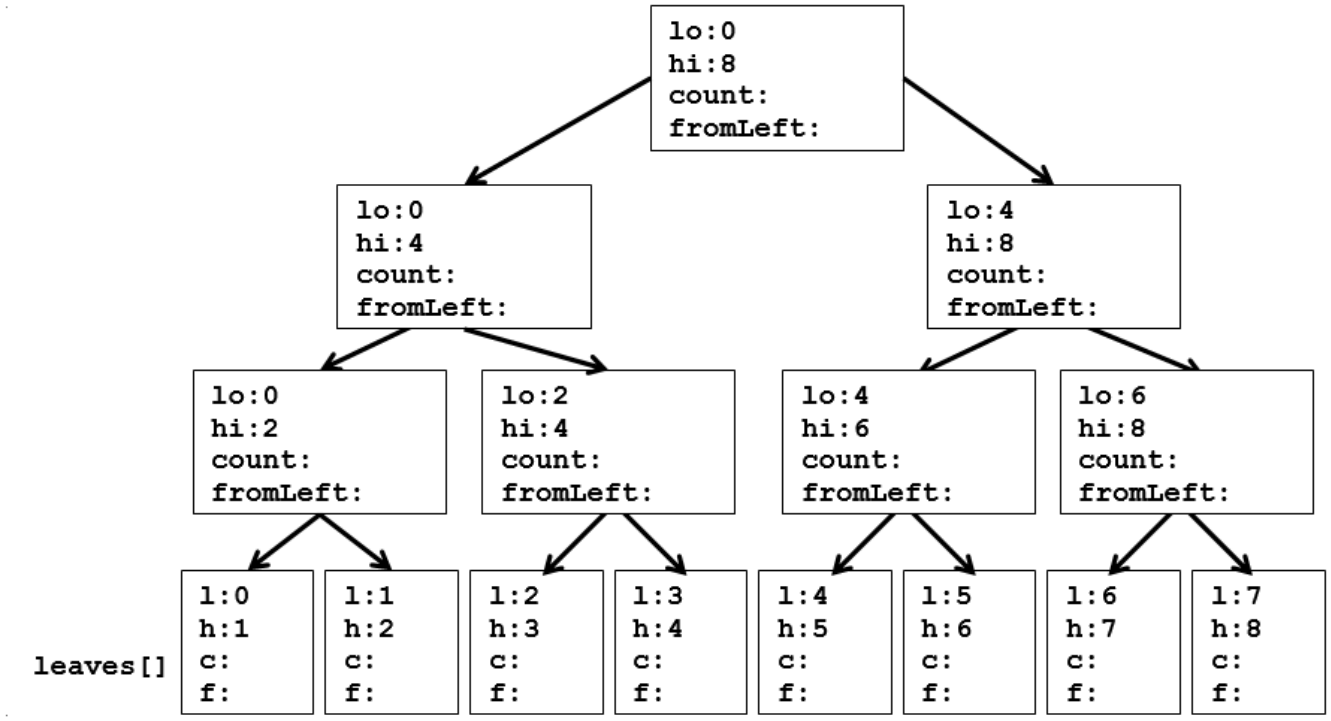
Given the following array as input, perform the parallel prefix algorithm to fill the output array with the **count of even numbers that appear in all of the cells to the left** in the `input` array (including the value contained in that cell). Do not use a sequential cutoff.

For example, for `input = {2, 12, 7, 3, 12, 5, -6, 0}`

`output` should be: `{1, 2, 2, 2, 3, 3, 4, 5}`.

a) Fill in the values for `count`, `fromLeft`, and the `output` array in the picture below given the following values for `input`. The `input` array has already been filled out for you. Note that problems b-e, on the next page, ask you to give the formulas you used in your calculation.

```
int[] input = {4, 7, 3, 5, 101, 3, 18, 3}
```



Index	0	1	2	3	4	5	6	7
input	4	7	3	5	101	3	18	3
output								

Q5 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) Give pseudocode for how you assigned a value to **`leaves[i].count`**

c) Give code for assigning **`p.left.fromleft.`**

`p.left.fromleft =`

d) Give code for assigning **`p.right.fromleft.`**

`p.right.fromleft =`

e) 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] =`

Q6: Sorting

a) What is the runtime of quicksort, given an input of **N identical values**?

O( )

b) True or False: if we replace the **maxHeap** with a **minHeap** in **in-place heap-sort**, the output will be sorted in **descending** order.

○ True ○ False

c) What is the runtime of mergesort, given an input of **N** values sorted in **descending** order?

O( )

d) Consider the following code to find an element in an array with comparable keys:

```
public static <K extends Comparable<K>> boolean find(K[] data, K key) {
  Arrays.sort(data);

  int lo = -1;
  int hi = data.length;
  while (lo + 1 != hi) {
    int mid = lo + (hi - lo) / 2;
    if (data[mid].compareTo(key) <= 0) {
      lo = mid;
    } else {
      hi = mid;
    }
  }
  return lo >= 0 && data[lo].equals(key);
}
```

The implementation of `Arrays.sort` is unknown to you.

Q6 continued on the next page…

Q6 continued:

Let **B(n)** be the best-case runtime and **W(n)** be the worst-case runtime of the above code, where **n** is the length of **`data`**. For each of the following statements, indicate whether it is **GUARANTEED** to be true.

i) B(n) is _____ to be Ω(n).

◯ guaranteed ◯ not guaranteed

ii) B(n) is _____ to be Ω(nlog(n)).

◯ guaranteed ◯ not guaranteed

iii) B(n) is _____ to be O(n).

◯ guaranteed ◯ not guaranteed

iv) W(n) is _____ to be Ω(nlog(n)).

◯ guaranteed ◯ not guaranteed

v) W(n) is _____ to be $O(n^2)$.

◯ guaranteed ◯ not guaranteed

Q7: Secret Question (0 pts)

Draw your TA drawing you drawing a proof of the following lemma:

$$\forall n \in \mathbb{N},\ n > 2 \Longrightarrow n! \nmid n^n$$

This is a blank page! Enjoy!

Summations

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

Logs:

1. $a^{\log_b(c)} = c^{\log_b(a)}$
2. $\log_b(a) = \frac{\log_d(a)}{\log_d(b)}$
3. $\log_b(b) = 1$
4. $\log_b(1) = 0$
5. $b^{\log_b(n)} = n$
6. $\log_b(n \cdot m) = \log_b(n) + \log_b(m)$
7. $\log_b(\frac{n}{m}) = \log_b(n) - \log_b(m)$
8. $\log_b(n^k) = k \cdot \log_b(n)$