23su CSE332 Final 2

Full Name: Solution

Email Address (UW NetID): This_better_not_be_a_number **@uw.edu**

Instructions:

- The allotted time is 1 hour.
- Do not turn the page until the staff says to do so.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- This is a closed-book and closed-notes exam.
- You are NOT permitted to access electronic devices including calculators.
- You must put your final answer inside the box.
    - If you run out of space, indicate where the answer continues.
    - Try to avoid writing on the very edges of the pages as we scan the exams.
- Unless otherwise noted, any bounds must be the **worst-case**, **simplified** and **tight**.
- Unless otherwise noted, logs are base 2.
- Unless otherwise noted, all material is assumed as in lecture.
- For answers that involve bubbling in a ◯ or ☐, fill in the shape completely.
- 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! :-).**

This page has been intentionally left blank.

Q5: ForkJoin (10 pts)

a) (10 pts) In Java using the ForkJoin Framework, write code to solve the following problem:

Input: An array of `int`s
Output: Print the index of the first even number, or `-1` if no numbers are even.

Example:
For example, if the input array is `[1, 2, 3, 4]`, the program would print `1`, because the first even number is at index 1.

Notes:

- Do not employ a sequential cut-off: the base case should process 1 element (you may assume the input array will contain at least one int).
- Give a class definition, `MEIT` (i.e., MinEvenIndexTask), along with any other code or classes needed.
- Fill in the function `printMinEvenIndex` below.

Fill in the underlines in the function `printMinEvenIndex` below.

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

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

    public static void printMinEvenIndex(int[] input) {

        int i = pool.invoke(new MEIT(0, input.length, input));
        System.out.println("First even element: " + i);
    }

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

Write your class here:

```
public static class MEIT extends RecursiveTask<Integer> {
    // Fields go here



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


    public Integer compute() {
        if (hi - lo <= 1) {
            if (arr[i] % 2 == 0) { // arr[i] is even
                return i;
            }
            return -1; // arr[i] is not even
        }

        int mid = lo + (hi - lo) / 2;
        MEIT leftTask = new MEIT(lo, mid, arr);
        MEIT rightTask = new MEIT(mid, hi, arr);
        leftTask.fork();
        int rightResult = rightTask.compute();
        int leftResult = leftTask.join();
        return leftResult >= 0 ? leftResult : rightResult;
    }
```

Q6: Concurrency (4 pts)

Consider the following **thread-safe** implementation of `Stack` class below:

```
1   public class Stack {
2       // Spec:
3       // 0 <= index < array.length
4       // array != null
5       private int index = 0;
6
7       Stack(int capacity) {
8           array = (E[]) new Object[capacity];
9       }
10
11      synchronized boolean isEmpty() {
12          return index==0;
13      }
14
15      synchronized void push(E val) {
16          if(index == array.length)
17              throw new StackFullException();
18          array[index++] = val;
19      }
20
21      synchronized E pop() {
22          if(index == 0)
23              throw new StackEmptyException();
24          return array[--index];
25      }
26  }
```

a) (2 pts) Suppose we remove the `synchronized` keyword from `isEmpty()`. Pick all the possible concurrency-related issues this would cause:

■ Data Race	■ Race Condition	□ Deadlock	□ None

b) (2 pts) Suppose we instead remove the `synchronized` keyword from the `pop()` method. **Informally** describe a bad interleaving that could happen.

- Bad interleaving is a series of execution of threads that violates a program's specification
- One of our specification is that `0 <= index < array.length`
- Violated when `index == 1`,
  - 2 threads call `pop()`,
  - both passing the `if(index == 0)` check and then decrement `index` twice to -1
  - (optional) also causes `ArrayIndexOutOfBoundsException`

This page has been intentionally left blank.

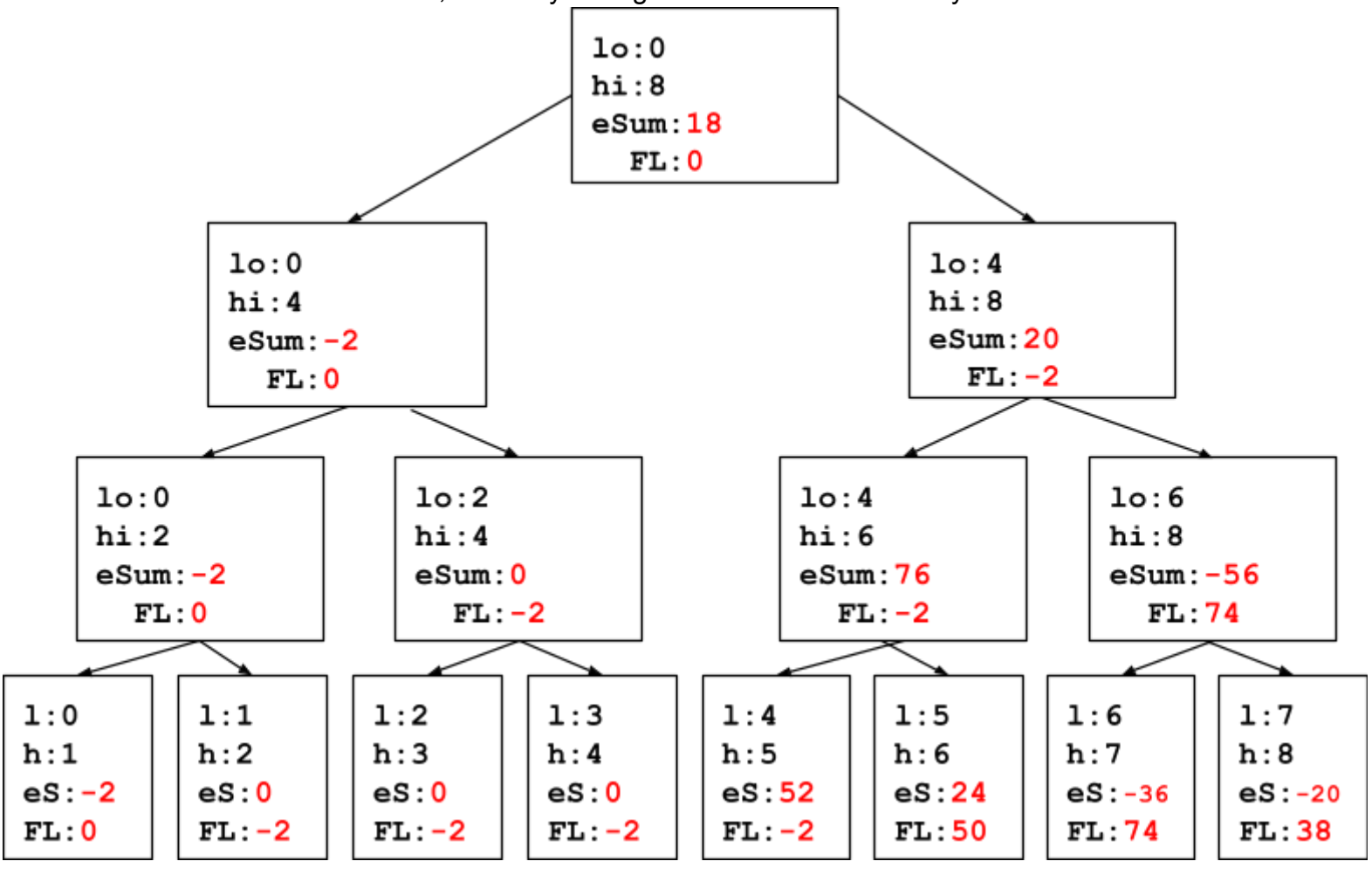
Q7: Parallel Prefix (10 pts)

Given the following array as input, perform a parallel prefix algorithm to fill the output array with the **sum of even numbers contained in all of the cells to the left** (including the value contained in that cell) in the input array.

Example:

Input: `[-3, 14, -1, 4, 5, 5, 6, 1]`
Output: `[0, 14, 14, 18, 18, 18, 24, 24]`

a) (5 pts) Fill in the values for `eSum`, `FL`, and the `output` array in the picture below. Note that later on, we ask you to give the formulas used in your calculation.



Index	0	1	2	3	4	5	6	7
input	-2	13	69	101	52	24	-36	-20
output	-2	-2	-2	-2	50	74	38	18

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) (2 pts) Give code for assigning `leaves[i].eSum`.

`leaves[i].eSum =` `(input[i] % 2 == 0) ? input[i] : 0;`

c) (1 pt) Give code for assigning `p.left.FL`.

`p.left.FL =` `p.FL;`

d) (1 pt) Give code for assigning `p.right.FL`.

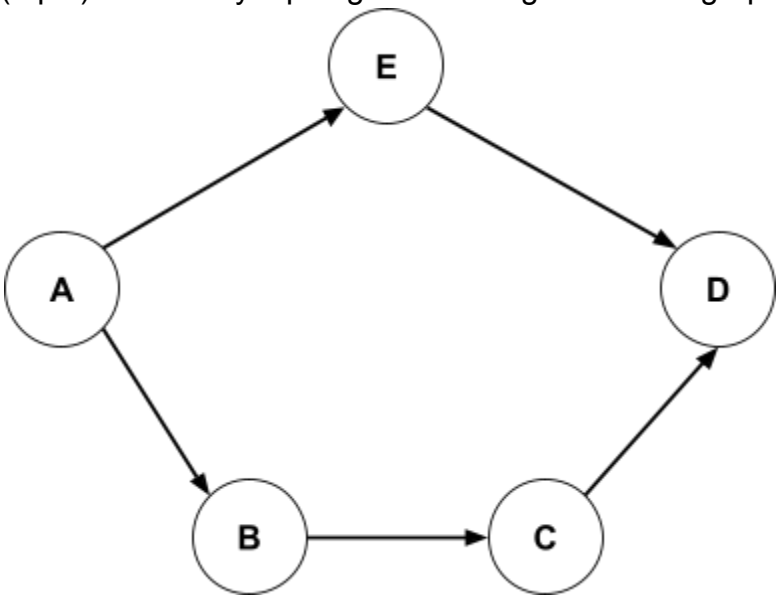
`p.right.FL =` `p.left.eSum + p.FL;`

e) (1 pt) Give code for assigning `output[i]`.

`output[i] =` `leaves[i].eSum + leaves[i].FL;`

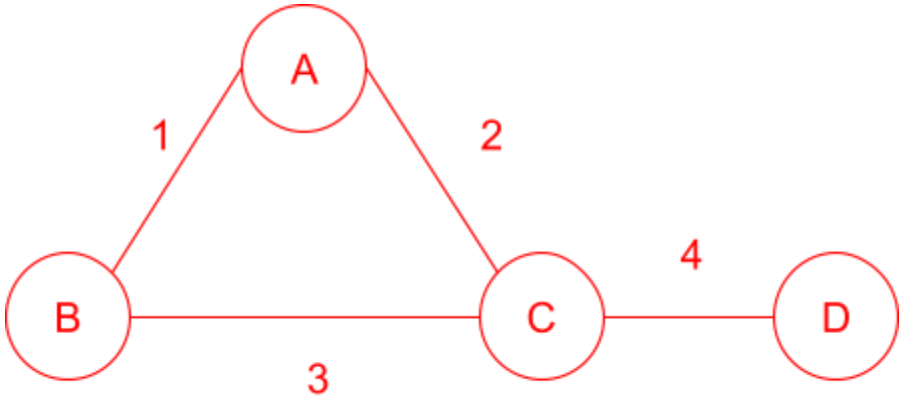
Q8: Graphs (7 pts)

a) (2 pts) How many topological orderings does this graph have?



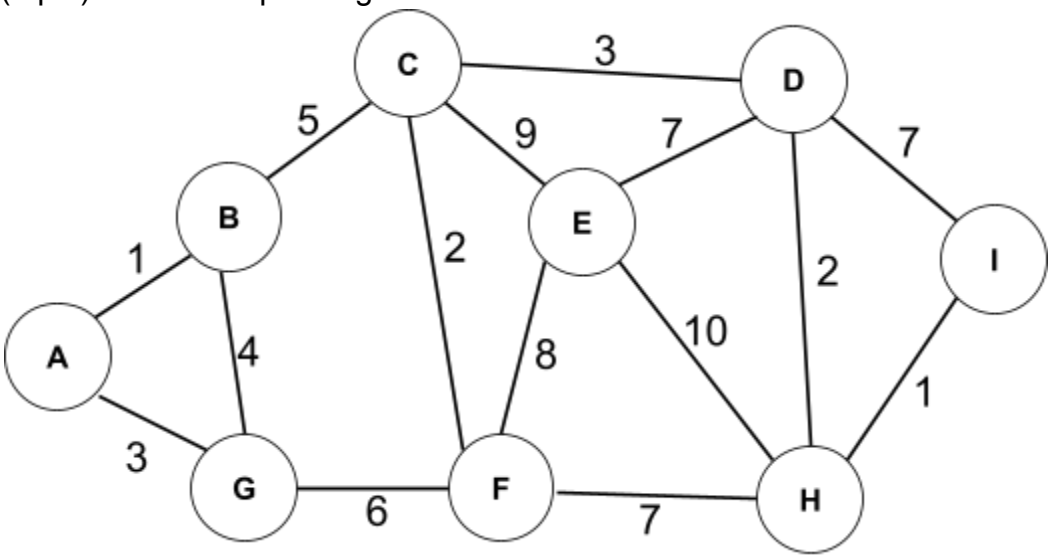
3

b) (3 pts) Draw a **Weighted**, **Undirected**, **Cyclic** Graph with **exactly 4** edges where the heaviest edge is contained in its MST. Each edge must have different weights.



Note that the edge weights had to be distinct. It's impossible for an edge to be in the MST if it's the heaviest edge in some cycle, so you can't just draw a "diamond".

c) (2 pts) Minimum Spanning Tree



Select all edges that are part of this graph's Minimum Spanning Tree.

☑ (A, B)	☑ (A, G)	☑ (B, C)	☐ (B, G)	
☑ (C, D)	☑ (C, F)	☐ (D, I)	☑ (D, E)	☑ (D, H)
☐ (E, H)	☐ (E, C)	☐ (E, F)	☐ (F, G)	☑ (H, I)

Q9: P/NP (9 pts)

a) (1 pt) "NP" stands for

Non-deterministic Polynomial

For the following problems, select ALL the sets each problem belongs to:

b) (1 pt) Determining if a chess move is the best move on an N x N board.

☐ **NP-Complete** ☐ **NP** ☐ **P** ■ **None of these**

c) (1 pt) Finding a cycle that visits every vertex exactly once.

■ **NP-Complete** ■ **NP** ☐ **P** ☐ **None of these**

For the following problems, decide whether the statement is True or False:

d) (1 pt) True or False: We know of an NP problem that is also undecidable.

○ **True** ● **False**

e) (1 pt) True or False: We can currently prove that there exists an NP problem that is not in P.

○ **True** ● **False**

f) (4 pts) Suppose you have a polynomial-time algorithm for 3-coloring. Describe why this means you also have a polynomial-time algorithm for 3-SAT.

Your answer should include the mention of a complexity class and a **general** explanation (preferably in bullet points) of how we could solve 3-SAT in polynomial time. You should not explicitly explain the details of the algorithms.

- 3-coloring is NP-complete/NP-hard and 3-SAT is NP/NP-complete
- NP-complete problems are reducible to one another
- 3-SAT is **polynomial** time reducible to 3-coloring
- Reduce 3-SAT problem to a 3-coloring problem and solve using the given polynomial-time 3-coloring algorithm

This page has been intentionally left blank.

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

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

This page has been intentionally left blank.