Final Exam

Winter 2026

Name Answer Key

Net ID _____ (@uw.edu)

Academic Integrity: You may not use any resources on this exam except for your one-page (front and back) reference sheet, writing instruments, your own brain, and the exam packet itself. This exam is otherwise closed notes, closed neighbor, closed electronic devices, etc.. The last three pages of this exam provide a list of potentially helpful identities and room for scratch work. Please detach those last two pages from the exam packet. No markings on these last two pages will be graded. Your answer for each question should fit in the answer box provided.

Instructions: Before you begin, **Put your name and UW Net ID at the top of this page.** Make sure that your name and ID are LEGIBLE. Please ensure that all of your answers appear within the boxed area provided.

Section	Max Points
Hashing	15
Sorting	12
Graphs	14
Parallelism	10
Concurrency	9
Extra Credit	(+2)
Total	60

Section 1: Hashing

(5 pts)**Question 1: Quadratic Probing**
Insert $22, 1, 13, 11, 24, 33, 35, 44, 55$ (in that order) into the open addressing hash table below. You should use the hash function $h(k) = k\%11$. In the case of collisions, use quadratic probing for collision resolution. Do not rehash. If an item cannot be inserted into the table, indicate this and continue inserting the remaining values.

Items that could not be inserted: 55

Index	Value
0	22
1	1
2	13
3	24
4	11
5	44
6	35
7	
8	
9	33
10	

(2 pts)**Question 2: Quadratic Probing**
When using quadratic probing, it is possible to have for an insert operation to be unsuccessful. How can we ensure all insertions will succeed?

make the array of prime length, keep the load factor under 0.5.

(2 pts)**Question 3: Quadratic Probing vs Linear Probing**
In what way is quadratic probing an improvement over linear probing?

Avoids primary clustering.

(6 pts)**Question 4: Hash Function Properties**
Below we provide a `equals()` method for an object called `MysteryObject`. None of the provided hash functions are considered to be a "good" hash function for that object because they do not possess one of these properties needed for a good hash function:

– **Consistent**: equal keys hash to the same integer

– **Effective**: the selection of an integer should behave has if it was done at random

Mention which property it lacks, and then use 1-2 sentences to describe why it does not have that property.

```
public class MysteryObject{
    public String field1;
    public String field2;
    public String field3;
    public boolean equals(MysteryObject other){
        return this.field1.equals(other.field1) && this.field2.equals(other.field2);
    }
}
```

[Problem continues on the next page]

1. ```
   public int hashCode(){
       return field1.hashCode();
   }
   ```

   Missing Property: Effective

   Justification:

   changing only field 2 results in a different object that hashes to the same value

2. ```
   public int hashCode(){
       return field1.hashCode() * field2.hashCode();
   }
   ```

   Missing Property: Effective

   Justification:

   swapping fields 1 and 2 results in a different object that hashes to the same value

3. ```
   public int hashCode(){
       return field1.hashCode() + 31*field2.hashCode() + 31*31*field3.hashCode();
   }
   ```

   Missing Property: Consistent

   Justification:

   field 3 is not relevant for determining equivalence, so we may have equal objects that hash to different values.


   ```

# Section 2: Sorting

(4 pts)**Question 5: Non-Comparison Sorting**

Suppose you have a list of $n$ non-negative integers, each smaller than $2n^{\sqrt{\log(n)}}$ (this is $n$ raised to the power of $\sqrt{\log(n)}$).

1. Describe the most efficient algorithm to sort your list. If you use bucket sort, mention how many buckets it requires. If you use radix sort, mention both the base that you will use as well as the length of the largest value when expressed using that base. (Hint: your answer will be either radix sort or bucket sort.)

   Radix sort. Use base $b = n$. Then the runtime is $O((n+b)\log_b m) = O(n\sqrt{\log n})$

2. Give a tight upper bound for the asymptotic runtime of your algorithm from the previous part.

   $O($ $n\sqrt{\log n}$ $)$

(4 pts)**Question 6: A Plethora of Unreliable Friends**

Suppose you had a sorted list of $n$ integers (but no bound on their size), and you split it into $n/k$ contiguous chunks (each of size $k$), and gave each chunk to one of your $n/k$ friends (you have a lot of friends) for safekeeping. Unfortunately, your plethora of friends are unreliable, and they each give you back an arbitrarily scrambled version of the list you gave them! You would like return your list to sorted order. You may assume that $n$ is a multiple of $k$.

For each subproblem below, we describe an algorithm for re-sorting your list, you must provide the running time of that algorithm.

1. Suppose you concatenate the chunks (in the correct order) your friends gave you, and then run insertion sort on the entire list. Give the asymptotic **worst-case** runtime of using this method to return your list to sorted order.

   $O($ $nk$ $)$

2. Suppose you concatenate the chunks (in the correct order) your friends gave you, and then run mergesort on the entire list. What would be the **worst-case** running time?

   $O($ $n\log n$ $)$

3. Suppose you mergesort each chunk, then concatenate them together (in the correct order). What would be the **worst-case** running time?

   $O($ $n \log k$ $)$

4. Suppose you concatenate the chunks (in the correct order) your friends gave you, and then run quicksort (on the entire list) using the first element as a pivot each time. What would be the **best-case** running time? As a hint, in the first iteration of the algorithm, the best case scenario is that the maximum element from the chunk is used as the pivot, which would then break the problem into two sublists: one of length $k-1$ and the other of length $n-k$.

   $O($ $\frac{n^2}{k} + n \log k$ $)$

(4 pts)**Question 7: Sports Draft Sorting**

For each scenario below, select the sorting algorithm property the situation most needs, then select the *fastest* sorting algorithm with that property. Select only from the options provided.

1. Sportsball players on college teams who wish to become professional sportsball players must enter the draft. In the draft, professional teams take turns selecting players to join their team. Suppose a team has a list of players who have entered the draft, and right now the players are sorted by their overall skill rating. The team wants to rearrange the list by the position they play so that they can see their ranking per position. Which algorithm should they use?

   **Property Options**: In Place, Stable, Adaptive, Online, Non-Comparison-Based.

   Property Needed: Stable

   **Algorithm Options**: Quick Sort, Insertion Sort, Heap Sort, Merge Sort.

   Algorithm Suggestion: Merge

2. Players with higher skill levels expect to have larger signing bonuses. Suppose a team wants to sign the cheapest player relative to their skill level when it's their turn to sign. To do this they will take the expected signing bonus of a player and divide that by the player's skill level, and round that value to the nearest integer. Suppose there are 400 players who have entered the draft, the maximum signing bonus is \$100,000, and the skill level is a score between 1 and 10.

   **Property Options**: In Place, Stable, Adaptive, Online, Non-Comparison-Based.

   Property Needed: Non-Comparison-Based

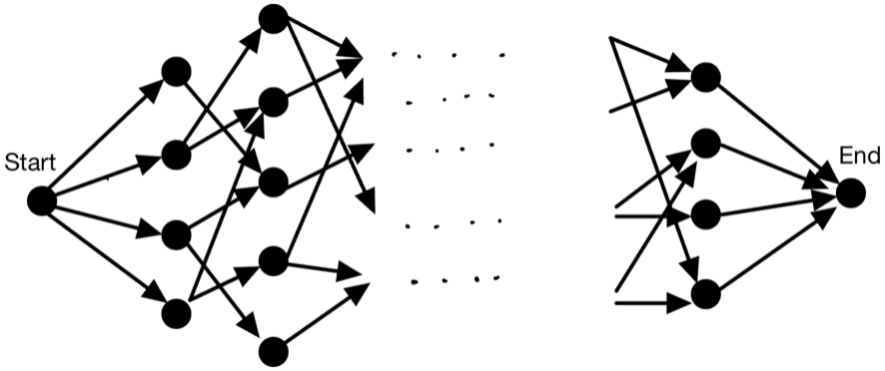
   **Algorithm Options**: Quick Sort, Insertion Sort, Heap Sort, Radix Sort.

   Algorithm Suggestion: Radix Sort

# Section 3: Graphs

(5 pts)**Question 8: A Particular Graph**

Suppose we have a weighted directed graph with $n$ vertices that we can decompose into layers, such that for every vertex in layer $i$, every of its out-edges points to some vertex in layer $i+1$. The first layer consists of a single start vertex, and the last layer consists of a single end vertex. See the below picture for an example. Although not included in the picture (for the sake of it not becoming too cluttered), we know **each edge** $e$ **has corresponding weight between** $-1000$ **and** $1000$.



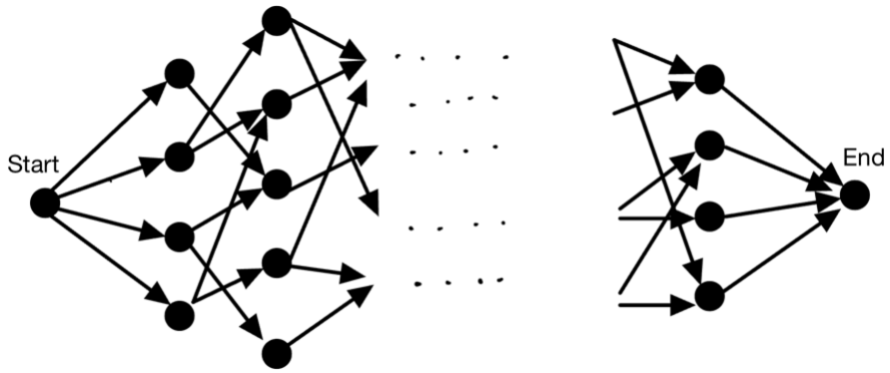

1. Dijkstra's algorithm will find the shortest path from Start to End.
   - ◯ Always
   - ● Sometimes
   - ◯ Never

2. Suppose we add 1000 to every edge. Then Dijkstra's algorithm will find the shortest path from Start to End.
   - ● Always
   - ◯ Sometimes
   - ◯ Never

3. Suppose we run DFS (starting from the start node). The last vertex be marked "done" is
   - ◯ A vertex in the layer just after Start (depends how we break ties)
   - ◯ A vertex in the layer just before End (depends how we break ties)
   - ● Start
   - ◯ End

[*Question continues on the next page.*]

The graph from the previous page has been copied below for your convenience.



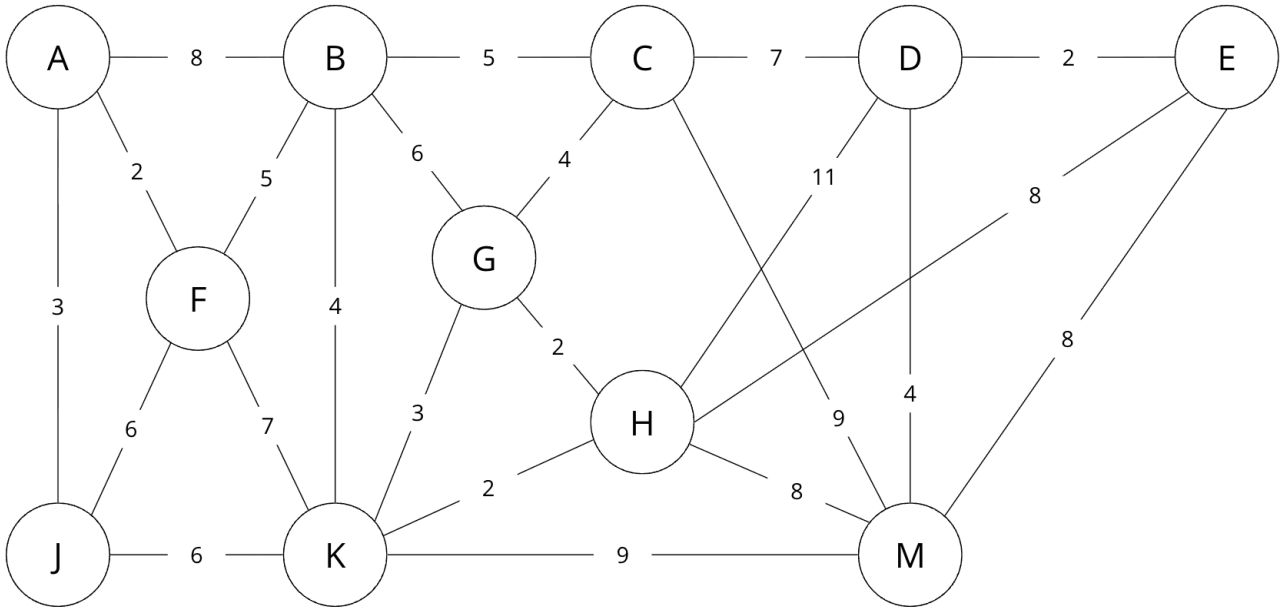

4. Suppose for this part that **each layer has exactly $k$ vertices** (except for Start and End). When we run BFS, which option describes the maximum size of the queue?
   - ◯ Every node ($n$)
   - ◯ Just the start node (1)
   - ◯ All of one layer ($k$)
   - ◯ All of one layer, except one node ($k-1$)
   - ◯ All of two layers ($2k$)
   - ● Two layers, except one node ($2k-1$)
   - ◯ All nodes except the start ($n-1$)
   - ◯ One node from each layer ($\frac{n-2}{k}+2$)

5. Suppose again that each layer has exactly $k$ vertices (except for Start and End). When we run DFS, which option describes the maximum number of vertices marked "visited" but not "done" at any point in the algorithm's execution?
   - ◯ Every node ($n$)
   - ◯ Just the start node (1)
   - ◯ All of one layer ($k$)
   - ◯ All of one layer, except one node ($k-1$)
   - ◯ All of two layers ($2k$)
   - ◯ Two layers, except one node ($2k-1$)
   - ◯ All nodes except the start ($n-1$)
   - ● One node from each layer ($\frac{n-2}{k}+2$)

(5 pts)**Question 9: Prim's Algorithm**

Use Prim's algorithm starting from node A to construct a Minimum Spanning Tree for the graph below. (In case you need a second try, this graph is duplicated in the scratch work at the end of the exam)




1. Which edges are added to the MST? (Fill in the circle to the left of each edge.)

| | | | | |
|---|---|---|---|---|
| A: | ○ A–B | ● A–F | ● A–J | |
| B: | ○ B–C | ● B–F | ○ B–G | ● B–K |
| C: | ● C–D | ● C–G | ○ C–M | |
| D: | ● D–E | ○ D–H | ● D–M | |
| E: | ○ E–H | ○ E–M | | |
| F: | ○ F–J | ○ F–K | | |
| G: | ● G–H | ○ G–K | | |
| H: | ● H–K | ○ H–M | | |
| J: | ○ J–K | | | |
| K: | ○ K–M | | | |

2. What is the total cost of the MST?

35

(4 pts)**Question 10: Guaranteed Edges**
Suppose we have an undirected graph $G$ with all of the following properties:

– $G$ is simple

– $G$ is connected

– $G$ has at least 4 vertices

– All edges in $G$ have unique weights

1. Select which of the edges below is the **least weight** edge which might not be present in the MST for $G$.
   - ◯ The edge with the smallest overall weight
   - ◯ The edge with the second smallest overall weight
   - ● The edge with the third smallest overall weight
2. Now give an example graph where its MST does not include that edge.

A graph with a triangle graph will work

# Section 4: Parallelism

(8 pts)**Question 11: ForkJoin**

For this question you will complete a parallel implementation of the following sequential method `fillSubtreeHeights` using the Java ForkJoin Framework. Consider this to be a method inside of a binary search tree class.

```
public class BST{
    public BSTNode root;
    ... //suppose other fields and constructors go here
    public class BSTNode{
        public int data;
        public int height;
        public BSTNode left;
        public BSTNode right;
    }
    public void fillSubtreeHeights(){
        fill(root);
    }
    public int fill(BSTNode curr){
        if(curr == Null){
            return -1;
        }
        int leftHeight = fill(curr.left);
        int rightHeight = fill(curr.right);
        curr.height = 1 + Math.max(leftHeight, rightHeight);
        return curr.height;
    }
}
```

This method populates the `height` field for each node in a binary search tree, then returns the `height` value of the current node. When finished, each node's `height` field will contain the height of the subtree rooted at that node (i.e. maximum number of edges from that node to a leaf).

On the next page we have an incomplete parallel implementation of `fillSubtreeHeights` using ForkJoin. In particular, we have provided:

– A BST class that has the `fillSubtreeHeights` method.

– Fields for a `FillTask` class

– The constructor for the `FillTask` class

– The signature of the `compute` method of `FillTask` including the base case.

And the code is missing:

– The body of the `fillSubtreeHeights` method, which should create an instance of `FillTask` and then call the `invoke` method of `ForkJoinPool`. (Starts on line 16)

– The class that `FillTask` should extend. (Line 20)

– The parallelized portion of the `compute` method. (Starts on line 32)

Complete our implementation by providing the missing code in the boxes following the code.

```
1  import java.util.concurrent.*;
2
3  public class BST {
4      private BSTNode root;
5      public static final ForkJoinPool POOL = new ForkJoinPool();
6      ... //suppose other fields and constructors go here
7
8      public class BSTNode{
9          public int data;
10         public int height;
11         public BSTNode left;
12         public BSTNode right;
13     }
14
15     public void fillSubtreeHeights(){
16         // Part 1 answer will go here
17     }
18 }
19
20 public class FillTask extends ??? { // Part 2 will replace the ???
21
22     public BSTNode curr;
23
24     public FillTask(BSTNode curr){
25         this.curr = curr;
26     }
27
28     public Integer compute(){
29         if(curr == null){
30             return -1;
31         }
32         // Your implementation of compute from Part 3 will go here.
33     }
34 }
```


Finish the code in the boxes below:

1. Implement the body of `fillSubtreeSums` in the box provided. The code you provide will be placed starting at line 16 of the code above

```
POOL.invoke(new FillTask(root));
```

2. Finish line 20 from the code above by filling the in the ??? with the class that `fillSubtreeSumsTask` should extend.

```
RecursiveTask<Integer>
```

3. Finally, finish the `compute` method. Your code will begin on line 32 above.

```
FillTask left = new FillTask(curr.left);
left.fork();
FillTask right = new FillTask(curr.right);
curr.height = 1 + Math.max(right.compute(), left.join());
return curr.height;
```

(2 pts)**Question 12: Amdahl's law**

Suppose I have a program that takes 100 seconds to run, but 80% of it can be parallelized with perfect linear speedup.

1. How long will the program run on 4 processors?

   40 seconds

2. What is the minimum number of processors needed to get the program to run in 30 seconds?

   8 processors

Section 5: Concurrency

(9 pts)**Question 13: Adding Edges**

The parts of this question reference the code below. This code implements an operation to add an edge in an undirected graph in parallel. The graph is represented by Node objects, and each Node object has a list of that node's neighbors. Assume all locks are reentrant.

```
1  public class ParallelGraph {
2      // Suppose other methods and fields are here.
3      public void addEdge(Node a, Node b){
4          List<Node> aList = a.getNeighbors();
5          List<Node> bList = b.getNeighbors();
6          synchronized(aList){
7              synchronized(bList){
8                  if (!aList.contains(b)) {
9                      aList.add(b);
10                     bList.add(a);
11                 }
12             }
13         }
14     }
15 }
```

For all subquestions, consider only two threads both running `addEdge` on the same `ParallelGraph` object.

1. If two threads are running `addEdge` there is a potential for deadlock. Describe an interleaving that results in deadlock.

   if two threads call addEdge with opposite choice of a and b, then each locks their respective aList before the other acquires bList.

2. Does the `addEdge` method have the potential for a data race? Answer "yes" or "no". no

3. Suppose we modify `addEdge` in the following way:

```
public void addEdge(Node a, Node b){
    List<Node> aList = a.getNeighbors();
    synchronized(aList){
        if (!aList.contains(b)) {
            aList.add(b);
            b.getNeighbors().add(a);
        }
    }
}
```

Is there potential for a data race with this modified code? yes

Is there potential for deadlock with this modified code? no

4. Suppose we modify `addEdge` in the following way:

```
public void addEdge(Node a, Node b){
    List<Node> aList = a.getNeighbors();
    List<Node> bList = b.getNeighbors();
    if(aList.size() < bList.size()){
        synchronized(aList){
            synchronized(bList){
                if (!aList.contains(b)) {
                    aList.add(b);
                    bList.add(a);
                }
            }
        }
    }
    else{
        synchronized(bList){
            synchronized(aList){
                if (!aList.contains(b)) {
                    aList.add(b);
                    bList.add(a);
                }
            }
        }
    }
}
```

Is there potential for a data race with this modified code? Yes

Is there potential for deadlock with this modified code? Yes

5. Suppose we instead modify `addEdge` in the following way (where each Node has an `id` field of type `int`, and all `id` values are unique):

```
public void addEdge(Node a, Node b){
    if(a.id < b.id){
        synchronized(a){
            synchronized(b){
                List<Node> aList = a.getNeighbors();
                List<Node> bList = b.getNeighbors();
                if (!aList.contains(b)) {
                    aList.add(b);
                    bList.add(a);
                }
            }
        }
    }
    else{
        synchronized(b){
            synchronized(a){
                List<Node> aList = a.getNeighbors();
                List<Node> bList = b.getNeighbors();
                if (!aList.contains(b)) {
                    aList.add(b);
                    bList.add(a);
                }
            }
        }
    }
}
```

Is there potential for a data race with this modified code? No

Is there potential for deadlock with this modified code? No

Extra Credit

(1 pts)**Question : Well Done!**
Overall, Nathan has felt that this has been an exceptional group of students, and is very proud of you all! More importantly, though, you should be proud of yourselves! Name one thing you accomplished this quarter that you're proud of.

...

(1 pts)**Question : or... well done?**
What is our professor's full name?

Nathan Brunelle

Identities

Nothing written on this page will be graded.

Summations

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \text{ for } |x| < 1$$

$$\sum_{i=0}^{n-1} = \sum_{n}^{i=1} = n$$

$$\sum_{i=0}^{n} i = 0 + \sum_{n}^{i=1} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$$

$$\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$$

$$\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$$

Logs

$$x^{\log_x(n)} = n$$

$$\log_a(b^c) = c \log_a(b)$$

$$a^{\log_b(c)} = c^{log_b(a)}$$

$$\log_b(a) = \frac{\log_d(a)}{\log_d(b)}$$

Scratch Work

Nothing written on this page will be graded.