Final Exam

Autumn 2024 Sample

Name _____

Net ID _____ (@uw.edu)

Academic Integrity: You may not use any resources on this exam except for writing instruments, your own brain, and the exam packet itself. This exam is closed notes, closed neighbor, closed electronic devices, etc.. The last two pages of this exam provide a list of potentially helpful identities as well as room for scratch work (respectively). 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 must 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
Asymptotic Analysis	14
Pre-Midterm Data Structures	14
Sorting	11
Graphs	13
Parallelism	17
Concurrency	13
Extra Credit	(+2)
Total	82

Section 1: Asymptotic Analysis

(4 pts)**Question 1: Asymptotic Analysis of Code**

Give a simplified Θ bound on the best and worst case running times for the given code. (By simplified we mean it should contain no constant coefficients or non-dominant terms.)

```
int doStuff(List<Integer> numbers){
    int n = numbers.size();
    int count = 0;
    if(n < 150){
        for (int i = 0; i < n; i++){
            for (int j = i; j < n; j++){
                count++;
            }
        }
    }
    else{
        for (int i = 0; i < n; i++){
            count += n;
        }
    }
    return count;
}
```

(a) Best Case: $\Theta($ $\quad$ $)$

(b) Worst Case: $\Theta($ $\quad$ $)$

(6 pts)**Question 2: Which is larger?**

For each pair of functions $f(n)$ and $g(n)$ below, select the choice which characterizes the asymptotic relationship between f and g. Write the letter corresponding with your answer in the box provided.

1. $f(n) = n^2 \log(4n)$, $g(n) = n \log_2(4^n)$

   A. $f(n) \in \Theta(g(n))$
   B. $f(n) \in O(g(n))$ and $f(n) \notin \Theta(g(n))$
   C. $f(n) \in \Omega(g(n))$ and $f(n) \notin \Theta(g(n))$

2. $f(n) = n^{1.5}$, $g(n) = n \log(n)$

   A. $f(n) \in \Theta(g(n))$
   B. $f(n) \in O(g(n))$ and $f(n) \notin \Theta(g(n))$
   C. $f(n) \in \Omega(g(n))$ and $f(n) \notin \Theta(g(n))$

3. $f(n) = n$, $g(n) = \sum_{i=0}^{n} \frac{i}{2}$

   A. $f(n) \in \Theta(g(n))$
   B. $f(n) \in O(g(n))$ and $f(n) \notin \Theta(g(n))$
   C. $f(n) \in \Omega(g(n))$ and $f(n) \notin \Theta(g(n))$

Creative Commons BY-NC 4.0

(4 pts)**Question 3: Recurrence Relation**

The method given below determines whether the sum of values in a given list is odd. First, express the worst case running time of the code as a recurrence relation. Next solve that recurrence relation using any method of your choice and express the running time as a simplified Θ bound.

```
boolean oddSum(int[] nums){
    return oddSumRec(nums, 0, nums.length);
}
boolean oddSumRec(int[] nums, int lo, int hi){
    if(hi-lo <= 0){return false;}
    if(hi-lo == 1){return nums[lo]%2 == 1;}
    if(hi-lo == 2){return (nums[lo] + nums[lo+1])%2 == 1;}
    int third = (hi-lo)/3;
    boolean left = oddSumRec(nums, lo, lo+third);
    boolean middle = oddSumRec(nums, lo+third, hi-third);
    boolean right = oddSumRec(nums, hi-third, hi);
    boolean odd = False;
    odd = (odd != left); // Note: != is the same as XOR
    odd = (odd != right);
    odd = (odd != middle);
    return odd;
}
```

1. Recurrence Relation:

2. Solved and simplified Θ bound:

Creative Commons BY-NC 4.0

Section 2: Pre-Midterm Data Structures

(5 pts)**Question 4: Heap**

Answer each question below as it relates to a 0-indexed binary min heap containing 65 items. 0-indexed means the root of the tree is at index 0 in its array representation.

1. What is the height of the tree (recall that a one-node tree has height 0)?

2. If we call percolate up on index 15, which index will we compare to?

3. If we call percolate down on index 15, which two indices will we compare to?

4. What is the smallest index which contains a leaf?

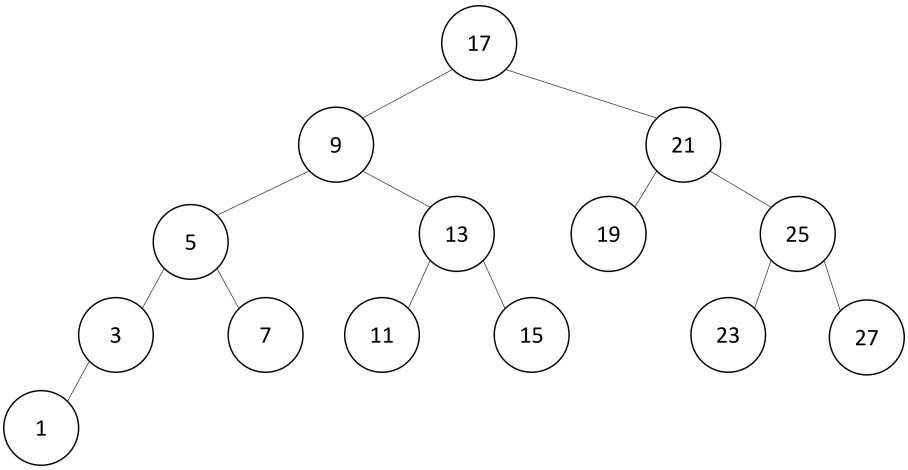
5. What index contains the smallest value?

Creative Commons BY-NC 4.0

(4 pts)**Question 5: AVL Tree**

Using the AVL Tree below, complete the table by indicating the type of AVL Tree rotation that would occur for each key inserted ("single", "double", or "none").

Each row should be considered completely independently (i.e. "reset" to the image between rows).



Key Inserted	Rotation Type (write "single", "double", or "none")
0	
2	
22	
26	

Creative Commons BY-NC 4.0

(5 pts)**Question 6: Double Hashing**

Insert 55, 15, 25, 28, 8, 48 (in that order) into the open addressing hash table below. You should use the primary hash function $h(k) = k\%10$. In the case of collisions, use double hashing for collision resolution where the secondary hash function is $g(k) = 1 + (k\%9)$. If an item cannot be inserted into the table, indicate this and continue inserting the remaining values.

Items that could not be inserted:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Creative Commons BY-NC 4.0

Section 3: Sorting

(4 pts)**Question 7: Sorting Olympics**
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. A total of 206 different nations have participated across 30 summer Olympics. The most gold medals ever one by a country in a single Olympics was the U.S., which won 83 gold medals in 1984. Which algorithm should we use to sort all 6,000 country-year pairs by the number of gold medals won by that country in that year?

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

   Property Needed: [ ]

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

   Algorithm Suggestion: [ ]

2. Suppose we had a list of all runners already sorted by their finish time in the 100 meter dash. A small number of runners incurred time penalties, so we need to sort the list again to adjust for those penalties. What algorithm should we use for this second sort?

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

   Property Needed: [ ]

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

   Algorithm Suggestion: [ ]

(2 pts)**Question 8: In-Place Sort**
Which of the following best matches the definition of an in-place sort? Write the letter of your choice in the box.

A. Items are re-ordered by swapping within the given list data structure

B. There is no randomness used in the sorting algorithm

C. The worst case running time matches the best case running time

D. Items that are already at the correct index will not be moved

E. If you re-run the algorithm on an already-sorted list, no items will change order

[ ]

Creative Commons BY-NC 4.0

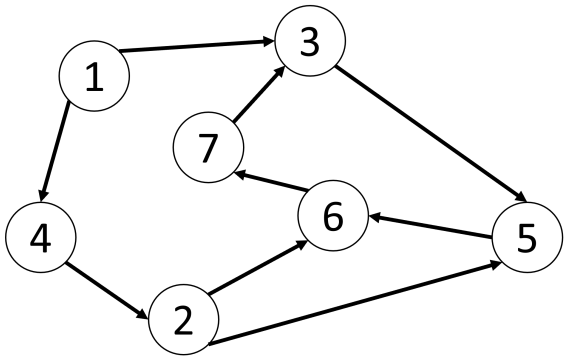
(3 pts)**Question 9: Quick Sort Runtime**
Using 1-2 sentences, explain why the method used to select the pivot for Quick Sort should not take more than linear time.

(2 pts)**Question 10: Decision Trees**
Recalling our decision tree argument to show a running time lower bound for comparison-based sorting algorithms, explain why the tree needed to have at least $n!$ leaves.

Creative Commons BY-NC 4.0

Section 4: Graphs

(2 pts)**Question 11: BFS**
For the graph below, list the nodes in an order that they might be removed from the queue in a BFS starting from node 1 (note that this is the same as the previous graph, but now undirected).



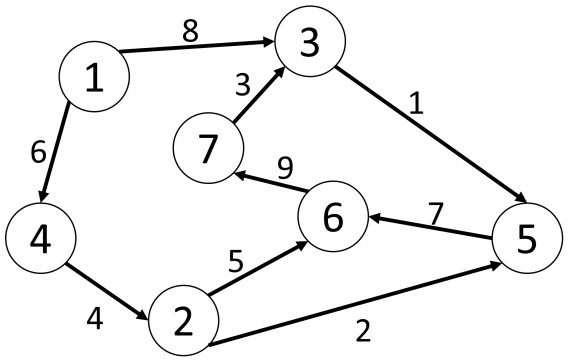
BFS Order:

(2 pts)**Question 12: DFS**
Using the same graph as the previous problem, list the nodes in a depth-first-search order starting from node 1. If there are multiple choices for the next node, you should always select the node with the smallest value first.

BFS Order:

(1 pt)**Question 13: Back Edge**
Identify the first back edge found by the DFS done in the previous problem.

(2 pts)**Question 14: Dijkstras**
For the graph below, list the nodes in an order that they might be removed from the priority queue when running Dijkstra's algorithm starting from node 1 (note that this is the same as the previous graph, but now with weights).

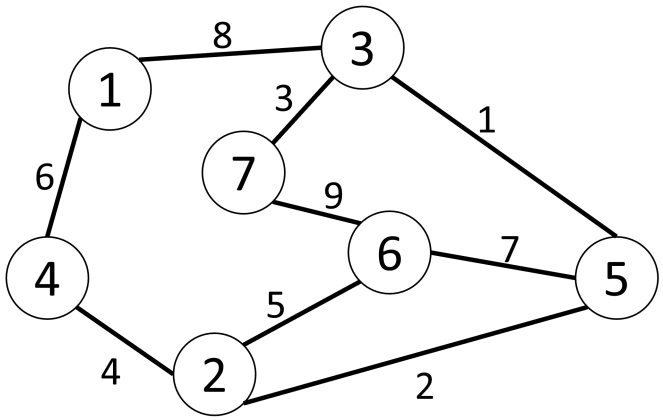


Dijkstra's Order:

Creative Commons BY-NC 4.0

(6 pts)**Question 15: MSTs**

The next 2 questions will relate to running minimum spanning tree algorithms (Kruskal's and Prim's) on the graph below (which is the same as the previous graph, but now undirected):



1. What are the weights of the first three edges added to the minimum spanning tree when running Kruskal's algorithm?

   First edge's weight: ____

   Second edge's weight: ____

   Third edge's weight: ____

2. What are the weights of the first three edges added to the minimum spanning tree when running Prims's algorithm starting with node 1?

   First edge's weight: ____

   Second edge's weight: ____

   Third edge's weight: ____

Creative Commons BY-NC 4.0

Section 5: Parallelism

(8 pts)**Question 16: ForkJoin**
For this question you will complete a parallel implementation of the following sequential method using the Java ForkJoin Framework.

```
static boolean lastEmpty(String[] arr){
    int last = -1;
    for(int i = 0; i < arr.length; i++){
        if (arr[i].length() == 0){
            last = i;
        }
    }
    return last;
}
```

This method returns the last index which contains the empty string from a given array of strings, or -1 if the empty string does not appear in the list. For example, given the array `["0", "1", "" , "3", "", "5"]` the method would return 4.

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

– A Client class that has the `lastEmpty` method.

– Fields for the `LastEmptyTask` class

– The constructor for the `LastEmptyTask` class

– The signature of the `compute` method as well as the sequential code that will run when the problem size is within the sequential cutoff.

And the code is missing:

– The body of the `lastEmpty` method, which should create an instance of `LastEmptyTask` and then call the `invoke` method of `ForkJoinPool`. (Starts on line 6)

– The class that `LastEmptyTask` should extend. (Line 10)

– The parallelized portion of the `compute` method. (Starts on line 31)

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

Creative Commons BY-NC 4.0

```
1  import java.util.concurrent.*;
2
3  public class Client {
4      public static final ForkJoinPool POOL = new ForkJoinPool();
5      public static int lastEmpty (String[] input) {
6          // Part 1 answer will go here
7      }
8  }
9
10 public class LastEmptyTask extends ??? { // Part 2 will replace the ???
11     String[] arr;
12     int hi;
13     int lo;
14
15     public LastEmptyTask(String[] arr, int lo, int hi){
16         this.arr = arr;
17         this.hi = hi;
18         this.lo = lo;
19     }
20
21     public Integer compute(){
22         if(hi-lo < 100){
23             int last = -1;
24             for(int i = lo; i < hi; i++){
25                 if (arr[i].length() == 0){
26                     last = i;
27                 }
28             }
29             return last;
30         }
31         // Your implementation of compute from Part 3 will go here.
32     }
33 }
```

Creative Commons BY-NC 4.0

Finish the code in the boxes below:

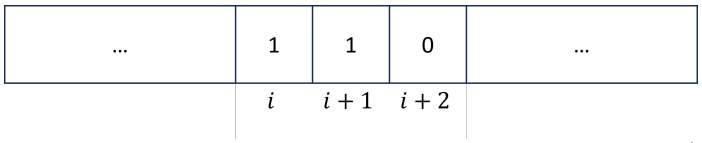
1. Implement the body of `lastEmpty` in the box provided. The code you provide will be placed starting at line 6 of the code above

2. Finish line 10 from the code above by filling the in the ??? with the class that `LastEmptyTask` should extend.

3. Finally, finish the `compute` method. Your code will begin on line 31 above.

(2 pts)**Question 17: Parallel Pack**

Suppose after the first stage of a parallel pack (i.e. filter) operation (that is, the map applying a boolean function to each index of the input array), you find that the value at index i is 1, at index $i+1$ is 1, and at index $i+2$ is 0. That is, the parallel map result appears like this:



Next, suppose that after the second stage of the same parallel pack operation (that is, the parallel prefix sum), you find that the value at index i is 10.

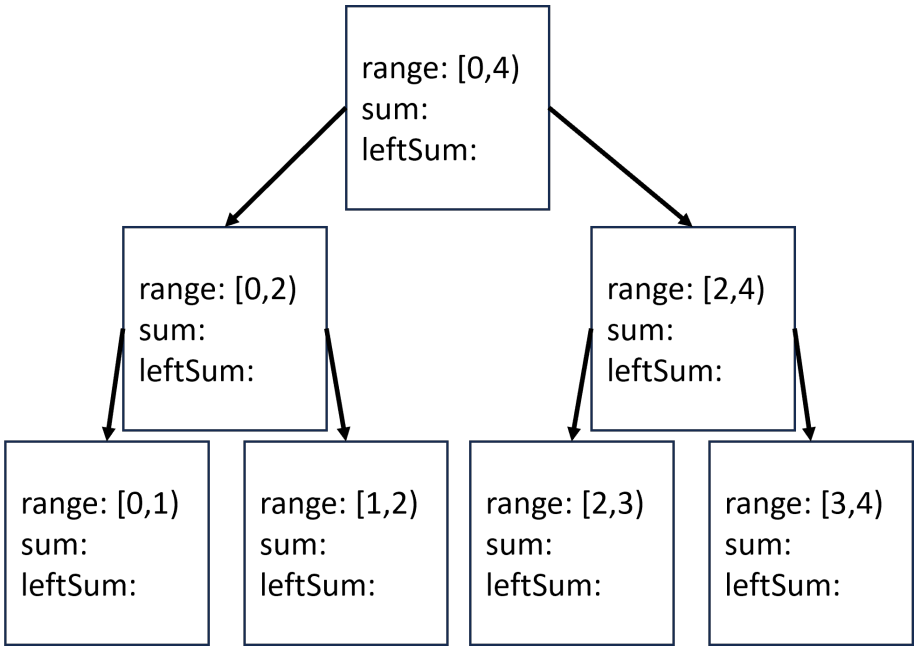
For each question below we give an index from the input array. If that item appears in the final output array, give its index. If does not appear, write "None".

1. At what index of the output array will you find the value that was at index $i+1$ of the input array?

2. At what index of the output array will you find the value that was at index $i+2$ of the input array?

(4 pts)**Question 18: Parallel Prefix**

Given the array `[3, 7, 0, 8]` as input, fill in the tree below as it would be completed by the parallel prefix sum algorithm.



Creative Commons BY-NC 4.0

(3 pts)**Question 19: Amdahl's Law**

Suppose we have a program in which a $\frac{3}{4}$ proportion can be parallelized with perfect linear speedup. Using $T_1 = 1$, answer the following using Amdahl's Law:

1. What is T_2?

2. What is T_4?

3. What is T_{10}?

Creative Commons BY-NC 4.0

Section 6: Concurrency

The remaining questions in this section use the classes below in a parallel implementation. Both relate to finding socks in a sock drawer. Two socks will match whenever they have the same color. To find a match we select a first sock at random, then repeatedly select a second sock at random until a match is found.

```
1  public static Random r = new Random();
2  public class Sock{
3      public String color;
4      public boolean isDirty = false;
5      public Sock(String color){
6          this.color = color;
7      }
8      public synchronized void wear(){
9          if(this.isDirty)
10             System.out.println("GROSS");
11         this.isDirty = true;
12     }
13 }
14 public class SockDrawer{
15     public List<Sock> drawer;
16     public SockDrawer(){
17         this.drawer = new ArrayList<>();
18     }
19     public Sock pickRandom(){
20         synchronized(drawer){
21             int choice = r.nextInt(0, drawer.size());
22             return drawer.get(choice);
23         }
24     }
25     public synchronized void putOn(Sock sock){
26         sock.wear();
27         drawer.remove(sock);
28     }
29     public void wearPair(){
30         Sock sock1 = pickRandom();
31         synchronized(sock1){
32             findMatch(sock1);
33         }
34     }
35     public Sock findMatch(Sock sock1){
36         Sock sock2 = pickRandom();
37         while(!sock2.color.equals(sock1.color) || sock1==sock2){
38             sock2 = pickRandom();
39         }
40         putOn(sock1);
41         putOn(sock2);
42     }
43 }
```

Note: The code is also provided on the last page of the exam, which you may detach for convenient reference.

(3 pts)**Question 20: Race Condition**

If `wearPair` is run sequentially, it is impossible for the code to print GROSS, regardless of the contents of the drawer. If two threads are both executing the `wearPair`, however, this is possible. Describe the drawer's contents and an interleaving that causes the code to print GROSS.

Is the error in this code a Data Race or a Bad Interleaving error? Write either DR or BI in the box to indicate your answer.

(2 pts)**Question 21: Deadlock**

This code additionally contains a potential for deadlock. Using about 3 sentences, explain how a deadlock could occur.

Creative Commons BY-NC 4.0

(8 pts)**Question 22: Deadlock and/or Race Condition**
Below we provide alternative implementations of the `wearPair` method.

For each implementation, consider two threads invoking the `wearPair` method. Indicate whether it is still possible for the code to print "GROSS" and whether there is still a potential for deadlock by writing "yes" or "no" in the corresponding box. Assume all code except for `wearPair` remains unchanged.

```
public void wearPair(){
    Sock sock1 = drawer.get(0);
    synchronized(sock1){
        findMatch(sock1);
    }
}
```

1. Deadlock? ____
2. Can Print GROSS? ____

```
public void wearPair(){
    Sock sock1 = drawer.get(drawer.size()-1);
    synchronized(sock1){
        findMatch(sock1);
    }
}
```

3. Deadlock? ____
4. Can Print GROSS? ____

```
public void wearPair(){
    synchronized(drawer){
        Sock sock1 = pickRandom();
        findMatch(sock1);
    }
}
```

5. Deadlock? ____
6. Can Print GROSS? ____

```
public void wearPair(){
    Sock sock1 = pickRandom();
    synchronized(sock1){
        drawer.remove(sock1)
        findMatch(sock1);
    }
}
```

7. Deadlock? ____
8. Can Print GROSS? ____

Extra Credit

(2 pts)**Question Extra Credit: What did you learn this quarter?**
Nathan's grandmother, Elouise, is 94 years old and has never used a computer before in her life. Summarize what you learned this quarter in a way that Elouise would appreciate.

Creative Commons BY-NC 4.0

Scratch Work

Nothing written on this page will be graded.

Creative Commons BY-NC 4.0

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

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

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

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

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

Creative Commons BY-NC 4.0

Section 6 Code

```
1  public static Random r = new Random();
2  public class Sock{
3      public String color;
4      public boolean isDirty = false;
5      public Sock(String color){
6          this.color = color;
7      }
8      public synchronized void wear(){
9          if(this.isDirty)
10             System.out.println("GROSS");
11         this.isDirty = true;
12     }
13 }
14 public class SockDrawer{
15     public List<Sock> drawer;
16     public SockDrawer(){
17         this.drawer = new ArrayList<>();
18     }
19     public Sock pickRandom(){
20         synchronized(drawer){
21             int choice = r.nextInt(0, drawer.size());
22             return drawer.get(choice);
23         }
24     }
25     public synchronized void putOn(Sock sock){
26         sock.wear();
27         drawer.remove(sock);
28     }
29     public void wearPair(){
30         Sock sock1 = pickRandom();
31         synchronized(sock1){
32             findMatch(sock1);
33         }
34     }
35     public Sock findMatch(Sock sock1){
36         Sock sock2 = pickRandom();
37         while(!sock2.color.equals(sock1.color) || sock1==sock2){
38             sock2 = pickRandom();
39         }
40         putOn(sock1);
41         putOn(sock2);
42     }
43 }
```

Creative Commons BY-NC 4.0