Name: __Sample Solution________________

Email address (UWNetID): ___________________________________

CSE 332 Autumn 2018 Final Exam

(closed book, closed notes, no calculators)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you **write down**, so show your work! Use only the data structures and algorithms we have discussed in class so far. Writing after time has been called will result in a loss of points on your exam.

Note: For questions where you are drawing pictures, please circle your final answer.

You have 1 hour and 50 minutes, work quickly and good luck!

Total: Time: 1 hr and 50 minutes.

Question	Max Points	Topic
1	8	Hashing
2	7	Graphs
3	11	More Graphs
4	10	|| Prefix
5	14	ForkJoin
6	12	Concurrency
7	22	Code & Sorting
8	6	Speedup
9	10	P/NP
Total	100	

1) [8 points total] Hash Tables

a) [4 pts] You are designing a **separate chaining hash table** and you are trying to decide between using a Binary Search Tree (BST) or a linked list as what each bucket should point to. List one pro and one con for each approach. Be specific and give answers as distinct from each other as possible.

BST:

Pro: If elements are inserted in random order, then you could expect a balanced tree and O(log N) runtime for find/insert/delete.

Con: BST has an "extra" pointer of space for each node. Keys also need to be comparable to insert them into a BST.

Linked List:

First describe how you would order the linked list. (circle one)

Ordered by increasing value OR most recently inserted item at the beginning

Pro: Both options use less memory than a BST because for each value there is space for only one pointer instead of two. For an "Insert at front LL" – can take advantage of temporal locality as most recently inserted element will be at the front of list.

Con: For both, the average case for find/insert/delete is O(N) (compared to an average case of O(log N) for a BST). For an "Insert at Front" LL, on a find for a value that is not present, you don't have the option of stopping searching once you reach a value larger than the one you are looking for.

b) [2 pts] Insert the following elements in this order: 7, 5, 24, 57, 16, 46, 4 into the Quadratic Probing hash table below. You should use the primary hash function h(k) = k%10. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining elements.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
46			**4**	**24**	**5**	**16**	**7**	**57**	

c) [2 pts] The Linear Probing hash table below already contains several elements, including two that have been lazily deleted. Insert the elements 77 and 33 (in that order) into the table below. You should use the primary hash function h(k) = k%10.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	11		23	~~(deleted)~~ **33**	35	56	~~(deleted)~~ **77**	88	

2) [7 points total] Graphs!

After moving into your new apartment in the Graf district, you want to make a sandwich to celebrate, but you need to buy ingredients first. You have a map of nearby buildings and their phone numbers.

a) [2 pts] First, you plan to call each building to find one that sells groceries. Given that edges between buildings in the Graf district are **all of length 1**, what simple algorithm should you use to efficiently find the nearest building that is a grocery store? (The name of the algorithm is sufficient. For full credit give the simplest algorithm possible.)

Breadth First Search (BFS)

b) [2 pts] It turns out that your phone doesn't work in the Graf district yet...so you plan to find the closest grocery store by first, just walk to all buildings 1 unit away, then walking to all buildings 2 units away, then all buildings 3 units away, and so on; going home after each iteration to rest, until you find a grocery store. What algorithm is this?

Iterative Deepening

c) [2 pts] As you prepare to start your journey, it occurs to you that maybe you could save time by planning a **single** route that visits each building in the Graf district **exactly once**, finally returning home! **Ignoring constant factors**, is this a good or bad idea, and **why**?

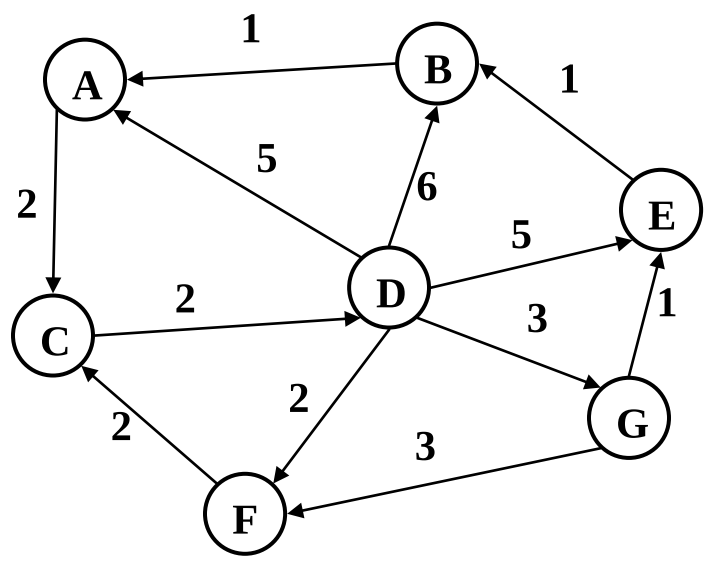
This is a bad idea in general. Finding a route that visits each building exactly once, and returns to your starting point is the Hamiltonian Circuit problem which is NP-complete. The iterative deepening approach is graph-linear.

-------------- Below here not related to the Graf district --

d) [1 pts] In class we discussed two data structures to represent graphs.
Give the name of the one that has a faster running time for the operation of determining if a particular edge exists.

________**Adjacency Matrix**________________

3) [11 points total] More Graphs! Use the following graph for this problem:



a) **[2 pts]** List a valid **topological ordering** of the nodes in the graph above (if there are no valid orderings, state why not).

No valid topological ordering exists, as there are multiple cycles in the graph (e.g. CDF).

b) **[5 pts]** Step through Dijkstra's Algorithm to calculate the single source shortest path from A to every other vertex. For full credit, you **must show all of your steps** in the table below **by crossing through Distance and Path values that are replaced by a new value.** Break ties by choosing the lowest 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.*

Vertex	Known	Distance	Path
A	~~F~~ T	~~∞~~ 0	-
B	~~F~~ T	~~∞~~ ~~10~~ 9	~~D~~ E
C	~~F~~ T	~~∞~~ 2	A
D	~~F~~ T	~~∞~~ 4	C
E	~~F~~ T	~~∞~~ ~~9~~ 8	~~D~~ G
F	~~F~~ T	~~∞~~ 6	D
G	~~F~~ T	~~∞~~ 7	D

c) **[1 pt]** In what order would Dijkstra's algorithm mark each node as *known*?

A, C, D, F, G, E, B

d) **[1 pt]** List the shortest path from A to G? (Give the actual path NOT the cost.

(A, C, D, G)

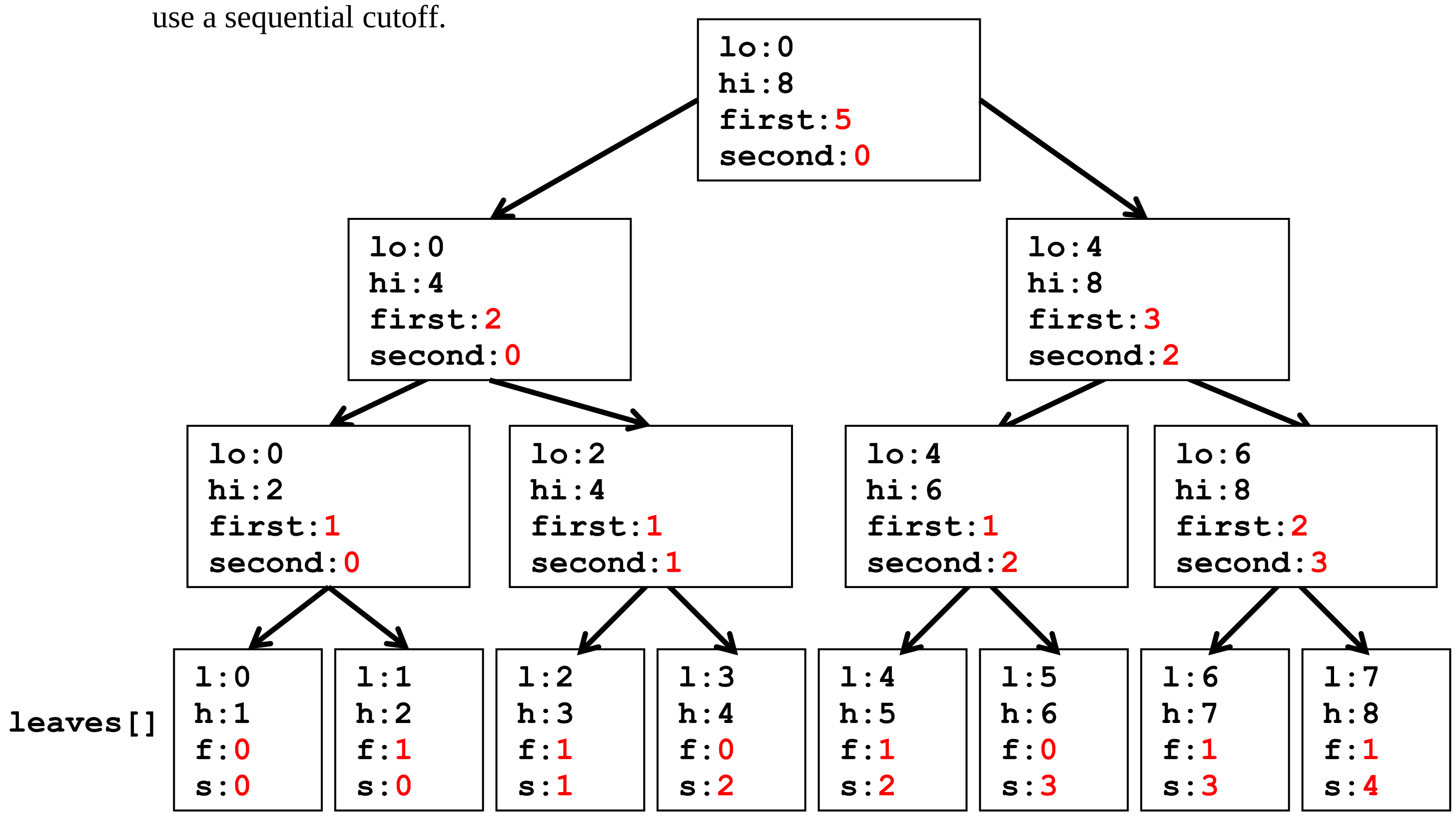
e) **[2 pts]** Is this graph strongly connected? (circle one) [✓ YES] NO

For any credit, briefly explain your answer.

There is a path from every vertex to every other vertex.

4) **[10 points] Parallel CountUnderTen:** Given the following array as input, perform the parallel prefix algorithm to fill the **output** array with the count of **only the values less than 10 contained in all of the cells to the left** (including the value contained in that cell) in the input array. Values >= 10 in the input array should not contribute to the count.

a) Fill in the values for `first` and `second` in the tree below. Use these variables as you wish: *in part b) you will explain how you used them.* The output array has been filled in for you. Do not use a sequential cutoff.



lo:0 hi:8 first:5 second:0
lo:0 hi:4 first:2 second:0
lo:4 hi:8 first:3 second:2
lo:0 hi:2 first:1 second:0
lo:2 hi:4 first:1 second:1
lo:4 hi:6 first:1 second:2
lo:6 hi:8 first:2 second:3
leaves[]
l:0 h:1 f:0 s:0
l:1 h:2 f:1 s:0
l:2 h:3 f:1 s:1
l:3 h:4 f:0 s:2
l:4 h:5 f:1 s:2
l:5 h:6 f:0 s:3
l:6 h:7 f:1 s:3
l:7 h:8 f:1 s:4

Index	0	1	2	3	4	5	6	7
Input	12	5	-8	34	6	10	2	7
Output	0	1	2	2	3	3	4	5

b) Give formulas for the following values where `p` is a reference to a tree node *other than a leaf 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 above.

`p.first = p.left.first + p.right.first`

`p.left.second = p.second`

`p.right.second = p.second + p.left.first`

`output[i] = leaves[i].second + leaves[i].first`

c) Describe how you assigned a value to: `leaves[i].first`

If `input[i]` is < 10, then assign value of 1, otherwise assign a value of 0.

5) [14 points] In Java using the ForkJoin Framework, write code to solve the following problem:

• **Input**: An array of `Strings`
• **Output**: Print the sum of the lengths of the strings AND the position of the longest string.

For example, if the input array is {“”, “hi”, “abcdef”, “a”}, the program would print 9 for the sum and 2 for the position of the string “abcdef”.

- Do **not** employ a sequential cut-off: **the base case should process one element**. (You can assume the input array will contain at least one string.)
- Give a class definition, `LengthSumTask`, **along with any other code or classes needed**.
- Fill in the function `printSumAndPosition` **below.**

You may not use any global data structures or synchronization primitives (locks).

a) Write the code.
b) Answer this: Is this a **map** or a [✓ **reduction**] **(circle one)**? **Why**?

We are reducing the problem from an array of size N down to 2 integers.

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

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

      // Print the sum of the lengths of the strings in input.
      // Also prints the position of the longest string.
      public static void printSumAndPosition (String[] input) {
         int sum, pos;
         // Your code here:

         Pair result =
           fjPool.invoke(new LengthSumTask(input, 0, input.length));
         sum = result.sum;
         pos = result.pos;









         System.out.println("Sum of lengths is: " + sum);
         System.out.println("Position of longest string is:" + pos);
      }
}
```

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

Don't forget to answer
b) on the previous page!

5) (Continued) Write your class(es) on this page.

```
public class Pair {
    public int sum;
    public int pos;
}

public class LengthSumTask extends RecursiveTask<Pair>{
     String[] array;
     int lo;
     int hi;


     public LengthSumTask(String[] array, int lo, int hi) {
          this.lo = lo;
          this.hi = hi;
          this.array = array;
     }

     protected Pair compute() {
         Pair result = new Pair();
         if (hi - lo < 2) {
          result.sum = array[lo].length();
          result.pos = lo;
         } else {
          int mid = lo + (hi - lo) / 2;
          LengthSumTask left = new LengthSumTask(array, lo, mid);
          LengthSumTask right = new LengthSumTask(array, mid, hi);

          left.fork();
          Pair rightResult = right.compute();
          Pair leftResult = left.join();

          result.sum = rightResult.sum + leftResult.sum;
          if (array[rightResult.pos].length() >
                              array[leftResult.pos].length()) {
              result.pos = rightResult.pos;
          } else {
              result.pos = leftResult.pos;
          }
         }
         return result;
     }

}
```

6) [12 points] Concurrency: The `BubbleTea` class manages a bubble tea order assembled by multiple workers. Multiple threads could be accessing the same `BubbleTea` object. Assume the `Stack` objects ARE THREAD-SAFE, have enough space, and operations on them will not throw an exception.

```
public class BubbleTea {
      private Stack<String> drink = new Stack<String>();
      private Stack<String> toppings = new Stack<String>();
      private final int maxDrinkAmount = 8;
      ReentrantLock drinkLock = new ReentrantLock();


      // Checks if drink has capacity
      public boolean hasCapacity() {

            return drink.size() < maxDrinkAmount;

      }

      // Adds liquid to drink
      public void addLiquid(String liquid) {
            drinkLock.acquire();
            if (hasCapacity()) {

                  if (liquid.equals("Milk")) {

                        while (hasCapacity()) {

                              drink.push("Milk");
                        }

                  } else {

                        drink.push(liquid);

                  }

            }
            drinkLock.release();
      }

      // Adds newTop to list of toppings to add to drink
      public void addTopping(String newTop) {

            if (newTop.equals("Boba") || newTop.equals("Tapioca")) {

                  toppings.push("Bubbles");

            } else {

                  toppings.push(newTop);

            }
      }
}
```

6) (Continued)
a) Does the `BubbleTea` class above have (circle all that apply):

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

If there are any problems, give an example of when those problems could occur. Be specific!

Assuming stack is thread-safe, a race condition still exists. If two threads attempt to call `addLiquid()` at the same time, they could potentially both pass the `hasCapacity()` test with a value of 7 for `drink.size()`. Then both threads would be free to attempt to push onto the drink stack, exceeding `maxDrinkAmount`.

Although this is not a data race, since a thread-safe stack can't be modified from two threads at the same time, it is definitely a bad interleaving (because exceeding `maxDrinkAmount` violates the expected behavior of the class).

b) Suppose we made the `addTopping` method **`synchronized`**, and changed nothing else in the code. Does this modified `BubbleTea` class above have (circle all that apply):

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

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Be specific!

Assuming stack is thread-safe, a race condition still exists as described above.

This change does reduce the effective concurrency in the code, however, so it actually makes thing slightly worse.

c) Modify the **code on the previous page** to use locks 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.** Create locks as needed. Use any reasonable names for the locking methods you call. **DO NOT use `synchronized`**. You should create re-entrant lock objects as follows:

```
ReentrantLock lock = new ReentrantLock();
```

d) Clearly circle all of the **critical sections** in your **code on the previous page**. DON'T FORGET

7) [22 points] Code Analysis & Sorting - A friend of yours has collected an array of delicious recipes to try baking. Each line of the recipe contains either an "ingredient" (e.g. `"ingredient: eggs"`) or an actual step in the baking process (e.g. `"step: crack the eggs into the bowl"`). Recipes always have at least one ingredient and at least one step. A sample recipe with a total of 7 lines and 4 steps might look like this:

```
ingredient: butter
ingredient: sugar
step: mix the butter and sugar
ingredient: flour
step: Add flour
step: Mix well
step: Bake
```

Your friend wrote some code to sort the recipes by how many **steps** they have, so that they can try baking the recipes in order of difficulty, from easiest (fewest number of steps) to hardest (largest number of steps). Their (pseudo)-code is shown below:

```
// sorts the provided array of recipes in ascending order of difficulty
void confectionSort(recipes):
    for i from 0 to recipes.length - 1:
        for j from i to recipes.length - 1:
            if isEasier(recipes[j], recipes[i]):
                  swap(recipes[i], recipes[j])

// helper function that returns true iff recipeA has fewer steps than recipeB
boolean isEasier(recipeA, recipeB):
    stepsInA = 0
    stepsInB = 0

    for line in recipeA.lines:
        if line is a step:
            stepsInA++
    for line in recipeB.lines:
        if line is a step:
            stepsInB++
    return stepsInA < stepsInB
```

Your friend wants your help analyzing and improving the code. **For each of the following four options** give the worst case Big-Oh runtime of the `confectionSort` algorithm, in terms of ***R*** (the number of recipes), and ***L*** (the maximum number of lines in any single recipe). Keep all terms of ***R*** and ***L*** in your final answers (e.g. do not assume **R** > ***L***).

a) **ORIGINAL SORT**: [Code shown above]
Running time:

$O(R^2 * L)$

b) **CHANGE #1**: [Original `isEasier` function, NEW `confectionSort` function that uses a **merge-sort strategy**.]
Running time:

$O(R * \log (R) * L)$

7) (Continued)

c) **<u>CHANGE #2</u>**: [NEW Modified `isEasier` function, Original `confectionSort` function.]
Describe how you would modify the `isEasier` function to improve overall `confectionSort` running time. You are allowed to add additional fields to the `recipe` objects if needed. You do not need to show code.
Description of Change: Add a "steps" field to each recipe object. Modify `isEasier` so that it only computes the steps in a recipe if it has not already been calculated. Thus each recipe's number of steps is only ever calculated once, so the runtime for this work is O(R*L). The original `confectionSort` will still take time O(R^2).

Running time: *O(R*L + R^2)*

d) **<u>CHANGE #3</u>**: [Throw everything out and start over!]
After talking some more, your friend mentions that although recipes sometimes contain many ingredients, every recipe has **at most 10 steps**. You are overjoyed! Describe what sorting algorithm your friend **should** be using.
Description of Change/New algorithm: They should use a bucket sort with 10 buckets! We still have to calculate the number of steps for each recipe in time *O(R*L)*. But once we do that we can put the recipes into buckets in time *O(R)*, and then read back through the 10 bins, printing the recipes out in time *O(R)*.

Running time: *O(R*L)*

e) Is ORIGINAL `confectionSort` an in-place sort? <u>For any credit, briefly explain why.</u>

[✓ **<u>YES</u>**] NO

Explanation: **No more than O(1) additional space is allocated.**

f) Is ORIGINAL `confectionSort` a stable sort? <u>For any credit, briefly explain why.</u>

YES [✓ **<u>NO</u>**]

Explanation: **The way swapping is used here could swap the relative position of two identical values, not by swapping the identical values with each other, but by moving one of them out of order when swapping with a different value. (e.g. try sorting the array [4, 4, 3])**

-------------- Below here not related to `confectionSort` --

g) **[3 points]** Give the ***<u>recurrence</u>*** for Quicksort (parallel sort & sequential partition) – worst case span: (Note: We are NOT asking for the closed form.)

```
T(n) = T(n-1) + O(n)
```

h) **[3 points]** Quicksort's partition step can also be parallelized. **<u>What</u>** is the big-O span of a single parallel partition? For full credit, **<u>explain why</u>** it has that span?

O(log N)
A partition consists of 2 pack operations (or the equivalent). A pack operation is O(log N) because it is made up of a map, followed by a parallel prefix, followed by another map (or the equivalent – some steps can be combined).

8) [6 points] Speedup

What ***fraction of a program must be parallelizable*** in order to get 5x speedup on 15 processors?

<u>**You must show your work for any credit**</u>. For full credit give your answer as a number or a simplified fraction (not a formula).

$$\frac{T_1}{T_{15}} = 5 = \frac{1}{S + \left(\frac{1-S}{15}\right)}$$

$$1 = 5 \cdot \left(S + \left(\frac{1-S}{15}\right)\right)$$

$$\frac{1}{5} = S + \left(\frac{1-S}{15}\right)$$

$$\frac{1}{5} = \frac{15S + 1 - S}{15}$$

$$1 = \frac{14S + 1}{3}$$

$$3 = 14S + 1$$

$$14S = 2$$

$$S = \frac{2}{14} = \frac{1}{7}$$

6/7 of the program must be parallelizable

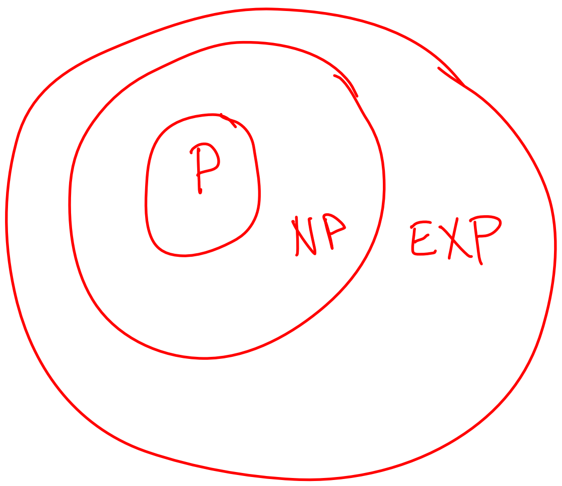
9) **[10 points] P, NP, NP-Complete**

a) **[1 point]** "NP" stands for ___**Nondeterministic Polynomial**________________________

b) **[2 points]** What does it mean for a problem to be NP-complete?

These problems are in NP (Given a candidate solution, we can verify whether the solution is correct in polynomial time.) and they are the "hardest" problems in NP. NP-complete problems can be reduced to other NP-complete problems in poly-time. We are pretty sure these problems do not have polynomial time solutions. If any one NP-complete problem could be solved in polynomial time, then all NP-complete problems could be solved in polynomial time.

c) **[4 points]** Draw a diagram demonstrating how (we are pretty sure) the sets P, NP, and EXP overlap/don't overlap with each other. Label each set clearly.



d) **[3 points]** Provide an example of ONE (if you list more than one you will get zero points) problem from each of the three sets below. Give a DIFFERENT problem for each set.

P: ____ **Euler Circuit** ____________

NP: __ **Hamiltonian Circuit** _______

EXP: ____ **NxN Chess** ____________