Name: ____Sample Solution_____

Email address (UWNetID): _____

CSE 332 Autumn 2019 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	9	Hashing
2	9	Graphs
3	6	More Graphs
4	10	|| Prefix
5	16	Concurrency
6	15	Sorting
7	10	P/NP
8	6	Speedup
9	14	ForkJoin
Total	95	

1) [9 points total] Hash Tables

a. **[3 pts]** Insert the following elements in this order: 37, 58, 19, 8, 17, 7 into the Double Hashing hash table below. You should use the primary hash function: `h(k) = k%10` and a secondary hash function: `g(k) = 1 + (k%6)`. 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
	8		17		7		37	58	19

g(8) = 1 + (8 % 6) = 1 + 2 = 3
g(17) = 1 + (17 % 6) = 1 + 5 = 6
g(7) = 1 + (7 % 6) = 1 + 1 = 2

b. **[2 pts]** The Quadratic Probing hash table below already contains several elements, including two that have been lazily deleted. Insert the elements 97 and 44 (in that order) into the 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
10	(deleted) **97**	21		14	(deleted) **44**	16	37	48	

c. **[2 pts]** You are designing a separate chaining hash table and you are trying to decide what each bucket should point to. The options are using a linked list where items are inserted at the front of the list or an AVL tree. **List one positive point for each approach when compared to the other approach**. Be specific about how that data structure gives you that behavior.

Linked List (where items are inserted at the front of the list): **A linked list where items are inserted at the front of the list will have better temporal locality than an AVL tree– items inserted recently will be quicker to find. Note that we still have to check if the item is already present on an insert since a hash table cannot have duplicates, so insert is still O(N) worst case. Also, this is not a "move to front list" as in p2. Linked lists do not need comparable keys, while AVL trees do.**

AVL Tree: **In the case that the load factor is high (there are several items per bucket) an AVL tree will have better worst case runtime on find/insert O(log N) when compared to the O(N) worst case behavior of the linked list.**

d. [2 pts] Give a tight big-O bound for the worst case runtime of a **Find** operation in a linear probing hash table **of size N^2** containing N elements. There have NOT been any deletions in the table. **Explain your what the worst case scenario is and why it would have this runtime.**

Worst Case Running time of Find:
O(N)

The worst case scenario is that all N items are clustered together contiguously in the hash table with no spaces between the N elements. This could happen due to all values hashing to the same initial location or otherwise. In this scenario the worst case is that the value you are trying to find is not actually present in the hash table but it hashes to the first location in the cluster and then must traverse all N elements in the cluster before finding an empty spot and determining that the element is not present.

2) [9 points total] Graphs!

a) **[4 pts]** You are given Kruskal's algorithm implemented using a priority queue and disjoint sets, as described in lecture. You are given a new implementation of disjoint sets with worst case running time of O(N) for find() and O(log N) for union(). Given that you must use this disjoint set implementation, what will be the worst case running time of Kruskal's? Give your answer as a tight Big-O bound in terms of V and E. **Explain how you got your answer briefly.**

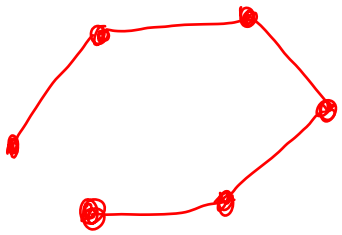
```
buildheap of edges                          // O(E)
while (numAccepted < V - 1) {               // E iterations at most
   get next lowest cost edge (u,v)          // O(log E) deletemin
   uset = find(u);                          // O(V) find
   vset = find(v);                          // O(V) find
   if (uset != vset) {                      // True V-1 times total
      numAccepted++;                        // O(1)
      union(uset, vset);                    // O(log V) union
   }
} So we get: O(E + E(log E + 2V) + V(1 + log V) )
```

Worst Case Running Time of Kruskal's with new disjoint sets:
O(Elog E + EV + Vlog V) or O(EV)

b) [2 pts] What is the minimum number of edges in a **connected** undirected **graph with 6 vertices**? **Draw a picture of your graph.**

Minimum Number of Edges:

5

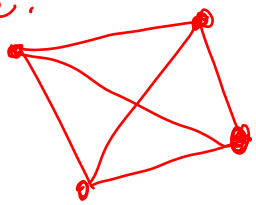


c) [2 pts] Give an EXACT number (**in terms of V**) for the minimum number of edges in a **complete** undirected graph without self-loops.

Minimum Number of Edges:

$\frac{V \cdot (V-1)}{2}$

Example:



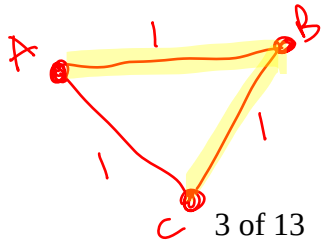
3+2+1 edges

$1 + 2 + 3 + \ldots + V-1$

$= \frac{(V-1) \cdot V}{2}$

d) **[1 pts]** True/False: The MST for a graph contains the shortest weight path between every pair of vertices. If true, explain briefly. If false, give a counterexample. Circle one: **TRUE** or **FALSE**

In this example the shortest path between A and C is the edge $\overline{AC}$ which is not included in the highlighted MST.



3) [6 points total] More Graphs!

a) **[2 pt]** For each of the following situations, provide an appropriate graph algorithm for solving the problem:

i. Given a social network graph, where nodes are people and edges connect people who are friends, find how many degrees of separation you are from a given target person. "Degrees of separation" is the smallest number of people you need to go through to be connected to the target person.

Algorithm: _____**Breadth First Search**_____

ii. Given a graph where the nodes are locations and the edges are the time needed to travel between locations, find the shortest path from the location "home" to the location "school".

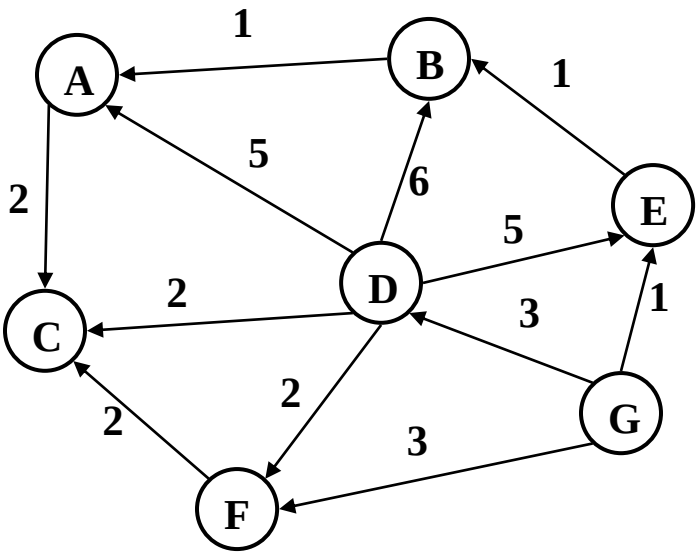
Algorithm: _____**Dijkstra's Single Source Shortest Path**_____

b) [2 pts] If you needed to calculate the **in-degree** of a single vertex in a graph, which representation would you prefer (circle one). Assume the basic representation of these we discussed in lecture:

adjacency matrix or **adjacency list**

In a couple of sentences describe WHY?

In an adjacency matrix, you only need to scan down a column to determine which vertices point to this one: O(V). In an adjacency list you would need to traverse the entire graph, counting how many times that particular vertex occurred in the linked lists: O(V + E).

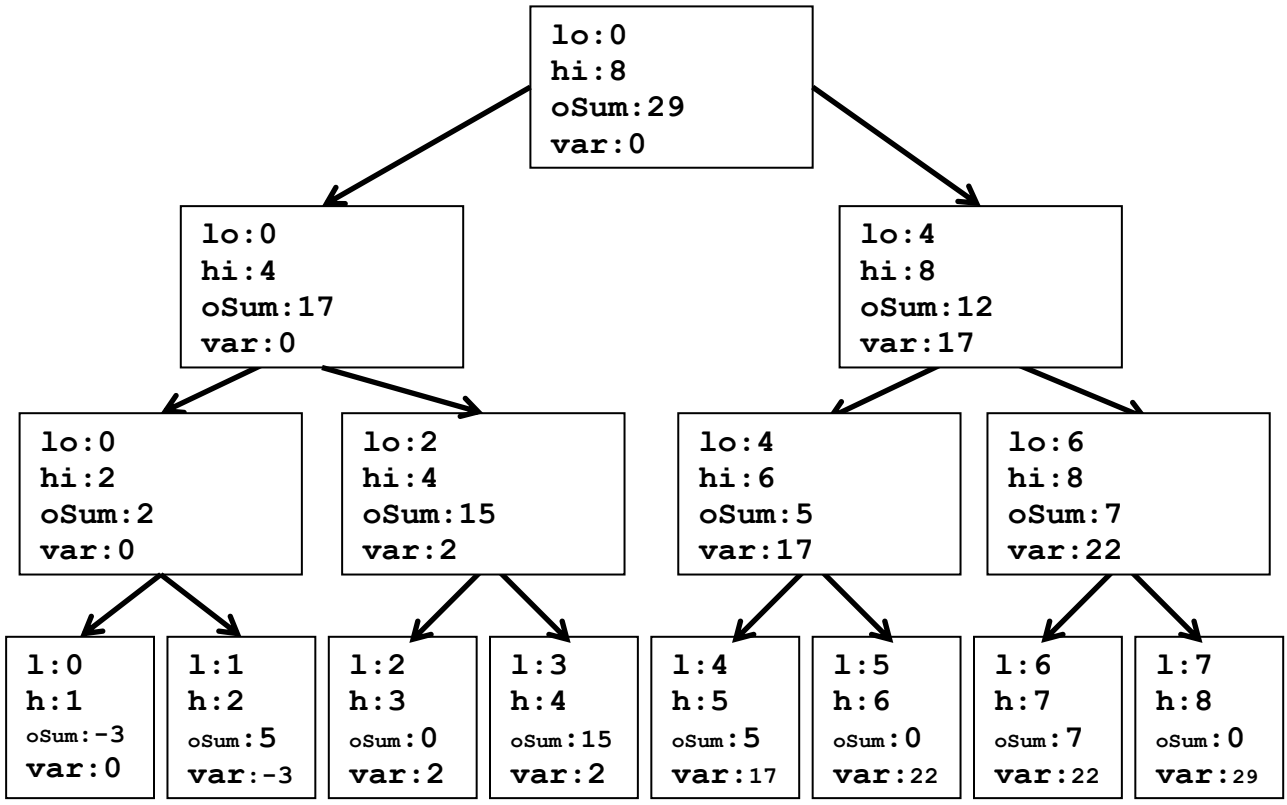


c) **[2 pts]** List a valid **topological ordering** of the nodes in the graph above (if there are no valid orderings, state why not).

G, D, F, E, B, A, C is one (F can appear anywhere after D)

4) [10 points] Parallel Prefix Sum of Odds:

a) Given the following array as input, perform the parallel prefix algorithm to fill the output array with the sum of only the odd values contained in all of the cells to the left (including the value contained in that cell) in the input array. Even values in the input array should not contribute to the sum (odd negative values should). Do not use a sequential cutoff.
For example, for input: {3, 14, -1, 4, 5, 5, 6, 1}, output should be: {3, 3, 2, 2, 7, 12, 12, 13}.
Fill in the values for **`oSum`** , **`var`** and the **`output`** array in the picture below.



Index	0	1	2	3	4	5	6	7
`input`	-3	5	4	15	5	10	7	2
`output`	-3	2	2	17	22	22	29	29

b) 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 above.

```
p.oSum = p.left.oSum + p.right.oSum

p.right.var = p.left.oSum + p.var

p.left.var = p.var

output[i] = leaves[i].oSum + leaves[i].var
```

c) Describe how you assigned a value to **`leaves[i].oSum`** .

```
if (input[i] % 2 == 1) then input[i] else 0
```

5**) [16 points total] Concurrency:** The `TimeMachine` class (code for entire class shown below) manages the CSE 332 staff's time machine. Multiple threads can access the same `TimeMachine` object.

```
1  public class TimeMachine {
2     private int now = 1985;
3     private int future = 2015;
4     private int energy = 100;
5
6  ReentrantLock energyLock = new ReentrantLock();
7  ReentrantLock futureLock = new ReentrantLock();
8
9     public boolean hasEnergy() {
10       energyLock.lock();
11       return energy >= 100; boolean result = energy >= 100;
12       energyLock.unlock(); return result;
13    .}
14
15    public void adjustEnergy(int charge) {
16       energyLock.lock();
17       if (energy + charge < 0 ) { // energy should never be negative
18          energyLock.unlock();
19          return;
20       }
21       energy = energy + charge;
22       energyLock.unlock();
23    }
24
25    public void setFuture(int newFuture) {
26       futureLock.lock();
27       future = newFuture;
28       futureLock.unlock();
29    }
30 }
```

a) **[4 pts]** Does the `TimeMachine` class as shown above have (circle **all** that apply):

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

Justify your answer. Refer to line numbers in your explanation. Be specific!

There are multiple data races. A thread could be in `hasEnergy` reading `energy` at line 11 while another thread is at line 21 in `adjustEnergy` writing `energy`. Two threads could also be at line 21 in `adjustEnergy` writing `energy`. Two threads could also be at line 27 in `setFuture` writing `future`. A data race by definition is a type of race condition.

5) (Continued)

b) [4 pts] We now add this method to the `TimeMachine` class:

```
28  public boolean backToTheFuture() {
29     energyLock.lock(); futureLock.lock();
30     if (!hasEnergy() && now != future) {
31        energyLock.unlock(); futureLock.unlock();
32        return false;
33
34     }
35
36     now = future;
37
38     energy = energy - 100;
39
40     System.out.println("Heading to:" + future + " Energy remaining:" + energy);
41     energyLock.unlock(); futureLock.unlock();
42     return true;
43
44  }
```

Does this modified `TimeMachine` class have (circle **all** that apply):

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

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example. Refer to line numbers in your explanation. Be specific!

This adds several new data races. A data race by definition is a type of race condition. Here are a few of the new data races:

- **A thread could be in `hasEnergy` reading `energy` at line 11 while another thread is at line 38 in `backToTheFuture` writing `energy`. Similarly a thread could be in `adjustEnergy` reading `energy` at line 17 or 21 while another thread is at line 38 in `backToTheFuture` writing `energy`.**
- **Two threads could also be at line 38 in `backToTheFuture` both writing `energy`, or one reading and one writing `energy` both on line 38. A thread could also be at line 40 in `backToTheFuture` reading `energy`, while another thread is at line 38 in `backToTheFuture` writing `energy`.**
- **Threads could be in `adjustEnergy` writing `energy` while a thread it reading `energy` at line 38 or 40 in `backToTheFuture`.**

- **A thread could be at line 27 in `setFuture` writing `future`, while a thread is at line 30 or line 36 or 40 in `backToTheFuture` reading `future`.**

c) **[8 pts]** Modify the **code above in part b) and 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 or extra locks.** 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();
```

6) [15 points total] Sorting

a) **[3 pts]** Give the ***recurrence*** for the PARALLEL MERGE discussed in lecture – **worst case span**. Note: We are NOT asking for the closed form. This is just the merge routine, *not full Mergesort*. **For any credit, explain all parts of your answer briefly.**

$T(n) = T\left(\frac{3}{4}n\right) + O(\log n)$ ← Binary search to find the median value in the "smaller" of the two pieces

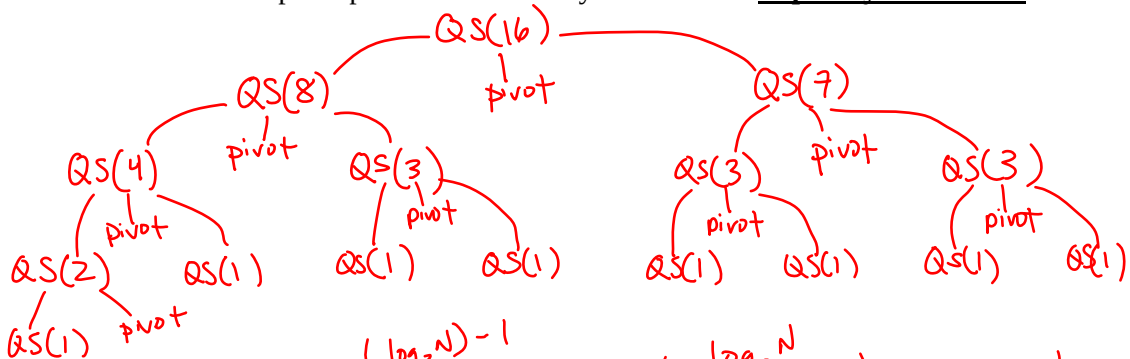
The worst case split is:

which comes from having 2 equal sized pieces of size $\frac{n}{2}$ finding the median of the larger piece (shown on the left) and determining (with binary search) that the entire smaller piece is either less than or greater than the median of the larger piece. Even though both calls to merge: $T\left(\frac{3}{4}n\right)$ and $T\left(\frac{1}{4}n\right)$ can be run in parallel, the $T\left(\frac{3}{4}n\right)$ term hides the $T\left(\frac{1}{4}n\right)$ term since it will finish last.

b) [4 pts] Give an exact expression in terms of N (closed form, not a recurrence, not Big-O) for the total number of times that **`quicksort`** and **`partition`** are called in a call to SEQUENTIAL QUICKSORT in the **best case**. Assume that partitioning is carried out by calling the helper method **`partition`**. Include the initial call to **`quicksort`** in your count. You should assume that N is a power of 2 and that no cutoff is used, **`quicksort`** will be called on a problem of size 1, **`partition`** will not be called on a problem of size 1. We want a simple expression without any summations. **Explain your answer.**

Exact Number of Times **`quicksort`** is called for problem of size N:

N



Exact Number of Times **`partition`** is called for problem of size N:

$\frac{N}{2}$

$\#\text{Calls to QS} = \sum_{i=0}^{(\log_2 N)-1} 2^i + 1 = \left(2^{\log_2 N} - 1\right) + 1 = N$

+1 For the one call on the next level down

$\#\text{calls to Partition} = \sum_{i=0}^{(\log_2 N)-2} 2^i + 1 = \left(2^{(\log_2 N)-1} - 1\right) + 1 = 2^{-1} \cdot 2^{\log_2 N} = \frac{1}{2} \cdot N$

Every element of the input will eventually correspond to either a **pivot** or a call to **QS(1)**. Every node in the call tree will either be a **leaf** (a call to QS(1)) or an **internal node** that corresponds to a call to QS on a value > 1 (which calls **`partition`** and picks one pivot). Thus we have N nodes in our tree and N total calls to QS. In the best case for Quicksort, for N = a power of 2, we will always have a shape like above: a perfect tree with one extra node at the next level down.

6) **(Continued)**

c) **[2 pts]** Give a tight Big-O bound for the running time of **insertion sort** if it *happened to be given an array of size N that was already sorted from largest to smallest*, and we wanted it sorted from smallest to largest. For any credit explain your answer.

Running time:

$O(N^2)$

Insertion sort examines the item immediately after the sorted portion of the array and inserts it into the appropriate location in the sorted part of the array. It does this by determining where it needs to go within the already sorted portion of the array and then moving some of these sorted items to the right if needed to make space for it. The worst case scenario is that on the kth iteration of the loop, k-1 items need to be scooted to the right by one space. This occurs when you are inserting an item that is smaller than any of the other currently sorted items. For an array that is sorted from largest to smallest, this will occur on every single iteration! Thus the number of items that need to be moved are 1 + 2 + 3 + 4 + … + N-1 which is $O(N^2)$.

d) **[3 pts]** Is Sequential Quicksort sort in-place? Circle one: **YES** NO

Explain your answer. Be specific – **refer to the Sequential Quicksort algorithm in particular**, do not just give a definition of in-place. Be sure to mention all relevant parts of the quicksort algorithm.

In-place means that only a constant amount of extra space is needed to perform the sort (e.g. no extra arrays of size N needed). Sequential quicksort is in-place because it does not use extra space beyond a few small scalar variables (e.g. pivot, temp etc.). The partitioning step in quicksort can be done in place as two pointers traverse the original array and swap values as needed. When quicksort is called twice, no extra copies of the original array are needed – the two calls to quicksort will be operating on separate portions of the original array. Thus no extra copies of the original array are ever needed. When returning from each call to quicksort no copies are made either. From lo to hi in the original array has been sorted on return from quicksort.

e) **[3 pts]** Ruth is considering using a single bucket sort to sort all of the students currently registered for CSE 332 (200 students) by their student number. Would you recommend she do this?

Circle one: YES **NO**

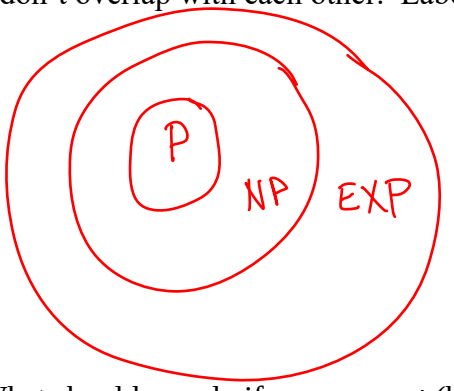
Explain your answer.

Student numbers consist of 7 digits. And while we may not need that entire range of buckets (from 0 to 9999999) the difference between the minimum student number and the max student number is likely to be much bigger than 200. Thus we would need a very large number of buckets. This would impact both space and time for the sort. The number of buckets would be much bigger than 200 and all but 200 of them would be empty – this would be a lot of wasted space, having poor memory behavior. When reading the sorted values out of the buckets we would need to traverse all of the buckets. Thus the runtime would be O(n + k) where n=200 and k = the number of buckets. Thus the number of buckets would dominate the overall running time. A radix sort would be a better choice.

7) **[10 points total] P, NP, NP-Complete**

a) **[1 pt]** "NP" stands for ___**Nondeterministic Polynomial**_____

b) **[2 pts]** 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.



c) **[2 pts]** What should you do if you **suspect** (but are not sure) a problem you are given is NP-complete, and **you need an answer to the problem in a reasonable amount of time**?

First try to establish that it is NP-complete. Part of this includes doing a reduction from a known NP-complete problem to your problem in polynomial time. (Another part includes showing that your problem is in NP, although that alone is not sufficient.)

AFTER you have shown that your problem is NP-complete, you can quit trying to find a polynomial time algorithm and instead use any of the techniques we use for NP-complete problems (e.g. approximation algorithms, solutions for a restricted version of the problem, heuristics).

d) **[2 pts]** If someone finds a polynomial time algorithm to solve the Traveling Salesman problem, name **one thing** that would likely be true (about the running times of algorithms):

Many things were accepted here. Examples include: P=NP, some other specific NP-complete problem or all NP-complete problems could be solved in polynomial time.

e) **[3 pts]** For the following problems, circle **ALL** the sets each problem belongs to:

Finding a minimum spanning tree in an undirected graph.	**NP**	**P**	NP-complete	None of these
Finding a path in a graph that begins and ends at the same vertex, and visits every vertex exactly once.	**NP**	P	**NP-complete**	None of these
Finding the shortest path from one vertex **to every other vertex** in a weighted directed graph.	**NP**	**P**	NP-complete	None of these

8) **[6 points] Speedup**

How many processors would you need to get 4x speedup on a program where 4/5 of the program is parallelizable? Is this possible?

<u>You must show your work for any credit</u>. For full credit give your answer as an integer (not a formula).

$$\text{Speedup on } P \text{ processors} = \frac{T_1}{T_P} = \frac{1}{S + \left(\frac{1-S}{P}\right)}$$

$$4 = \frac{1}{\frac{1}{5} + \left(\frac{1-\frac{1}{5}}{P}\right)}$$

$$4\left(\frac{1}{5} + \frac{\frac{4}{5}}{P}\right) = 1$$

$$\frac{1}{4} = \frac{1}{5} + \frac{4}{5 \cdot P}$$

$$\frac{1}{4} - \frac{1}{5} = \frac{4}{5 \cdot P}$$

$$\frac{5}{20} - \frac{4}{20} = \frac{1}{20} = \frac{4}{5 \cdot P}$$

$$80 = 5 \cdot P$$

$$P = \frac{80}{5} = 16$$

Yes, it would take 16 processors

9) [14 points] In Java using the ForkJoin Framework, write code to solve the following problem:

- **Input**: An array of positive `ints`.
- **Output**: Print the two most commonly occurring **ones place digits**. If there is a tie in how common digits are, the smallest digit should be considered more common.

Input array: {2005, 5, 4, 35, 44, 3} most common: 5, second most common: 4
Input array: {7, 6, 17, 3} most common: 7, second most common: 3
Input array: {13, 6, 7, 6003, 17, 766} most common: 3, second most common: 6

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

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

      // Prints the two most common digits occurring in the ones place.
      public static void printTopTwo (int[] input) {
         int top;
         int second;
         // Your code here:

         int[] res = fj.invoke(new TopTwoTask(0, input.length, input));

         top = 0;
         second = 1;

         for (int i = 0; i < 10; i++) {
            if (res[i] > res[top]) { // new top, move top to second
               second = top;
               top = i;
            } else if (res[i] > res[second]) { // only new second
               second = i;
            }
         }


         System.out.println("Most common ones place digit: " + top);
         System.out.println("Second most common ones place digit:" + second);
      }
}
```

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

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

```
public static class TopTwoTask extends RecursiveTask<int[]> {
   int lo, hi;
   int[] input;

   public TopTwoTask(int lo, int hi, int[] input) {
     this.lo = lo;
     this.hi = hi;
     this.input = input;
   }

   public int[] compute() {
      if (hi - lo <= 1) {
         int[] counts = new int[10];
         counts[input[lo] % 10] = 1;
         return counts;
      }

      int mid = lo + (hi - lo)/2;

      TopTwoTask left = new TopTwoTask(lo, mid, input);
      TopTwoTask right = new TopTwoTask(mid, hi, input);

      right.fork();
      int[] leftResult = left.compute();
      int[] rightResult = right.join();

      for (int i = 0; i < 10; i++) {
         leftResult[i] += rightResult[i];
      }
      return leftResult;
   }
}
```