Note: These problems are written by TAs and are not necessarily representative on the final exam. Note that problems have been organized by topic, with problems on the final unit appearing at the end.

Key: CW = code writing, CT = code tracing, SA = short answer, MC = multiple choice,

Unit: Data Structures and Efficiency

1. **Lists CW:** Write the mutating function `shuffleQueue(L, n)` that takes in a list of songs represented as strings and an integer n, and shuffles the input list L **with mutation** n times by choosing two random songs in the list and switching their locations. If the two songs randomly chosen on a given iteration are the same song, do not count the current iteration in the n shuffles.

2. **Recursion CW:** Write the **recursive** method `sumOfPositiveEvens(L)` that takes in a list L and returns a sum of all the even and positive integers in the list. For example, `sumOfPositiveEvens([1,2,3,4,5])` should return 6 and `sumOfPositiveEvens([-2,-4,-6,-8])` should return 0.

3. **Dictionaries CW:** Write a function `combineCovidDicts(dict1, dict2)` that takes in two dictionaries, dict1 and dict2, each of which contains keys that represent cities (strings) with the corresponding value being the percentage of individuals in that city with Covid (float). This function should output a result dictionary which combines the two inputs as specified below:
    a. If a city is key both dict1 and dict2, set that city as a key in your result dictionary with a value equal to the percentage for that city in dict2 - the percentage for that city in dict1
    b. If a city is only in dict2, then just set that city as a key in your result dictionary with a value equal to the percentage for that city in dict2
    c. If a city is only in dict1, then just set that city as a key in your result dictionary with a value equal to the percentage for that city in dict1

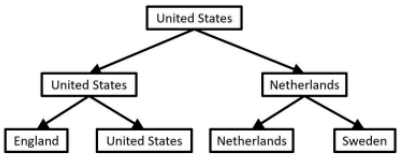
4. **Big-O SA:** What's the big O of the following function?

```
def mystery(n):
    result = 0
    start = 1
    while start <= n:
        result += start
        start *= 2
    for i in range(result * n):
        print("Hello")
```

   a. What is the Big-O complexity of this function if we only count the **bolded multiplication step**?
   b. What is the Big-O complexity of this function if we only count the **bolded print statement**?

5. **Trees CW:** We are back at it representing a tournament as a binary tree. Recall how the information from last time worked… ("United States" is misspelled "United Stated" on accident in a few places)

For example, the following bracket represents the last two rounds of the Women's World Cup in 2019.



In our binary tree dictionary format, this would look like:

```
t1 = { "value" : "United Stated",
       "left" : { "value" : "United Stated",
           "left"  : { "value" : "England", "left" : None, "right" : None },
           "right" : { "value" : "United States", "left" : None, "right" : None } },
       "right" : { "value" : "Netherlands",
           "left"  : { "value" : "Netherlands", "left" : None, "right" : None },
           "right" : { "value" : "Sweden", "left" : None, "right" : None } }
     }
```

Write the function `opponentsBeat(tree, team)` which takes a tournament bracket in (as a tree) and a team from that bracket (as a string), then returns a result list of all the opponents that that team beat in the bracket. Note that it should return an empty [ ] if the team beat no one and should be done **recursively**. Observe that since this is a bracket, it means every node that isn't a leaf must have two children since it takes two teams to play a game.

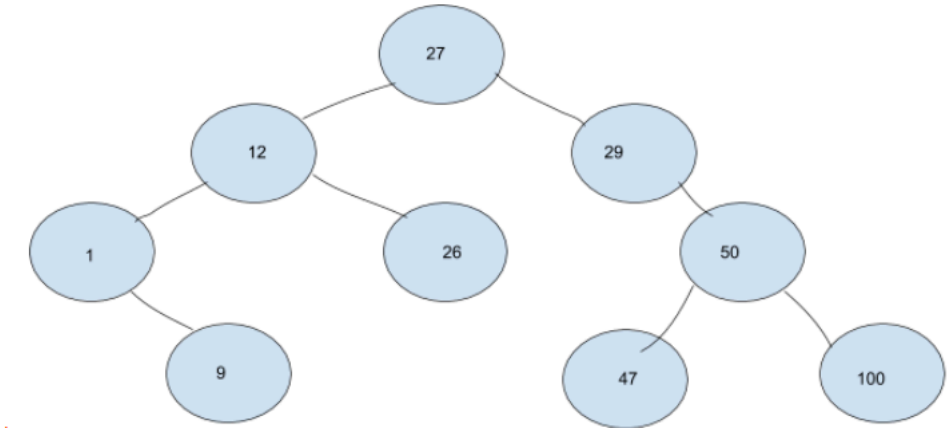
Hint: One solution involves having 1 base case that checks for a leaf (figure out why that is); observe that if the node is not a leaf, then the children of the node are the team itself and the opponent they beat.

Exs:
`opponentsBeat(t1, "United States")` → `["Netherlands", "England"]`
`opponentsBeat(t1, "England")` → `[ ]`

6. **Trees CT:** Consider the following function:

```
def oddValuesTree(tree):
    if tree == None:
        return []
    else:
        if tree["contents"] % 2 == 0:
            return (oddValuesTree(tree["left"]) +
                   oddValuesTree(tree["right"]))
        else:
            return ([tree["contents"]] +
                   oddValuesTree(tree["left"]) +
                   oddValuesTree(tree["right"]))
```

If the function is given the tree below, what should the return value be?

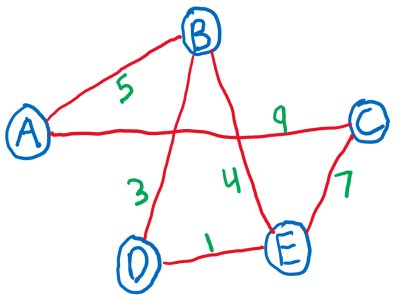


The function has now changed to this…

```
def oddValuesTree(tree):
    if tree == None: #base case
        return []
    else: #recursive case
        if tree["contents"] % 2 == 0:
            return (oddValuesTree(tree["right"]) +
                    oddValuesTree(tree["left"]))
        else:
            return (oddValuesTree(tree["right"]) +
                    oddValuesTree(tree["left"]) +
                    [tree["contents"]])
```

If we pass in the same tree, what is the return value now?

7. **Graphs CW:** Write a function `maxEdgeWeight(g)` that takes in a graph g and returns a list of the maximum weight edges for each node in the graph. Each edge is represented as a two-element list (the two nodes that form the edge). If there are no edges connected to a given node, don't add anything to the result list for that node.



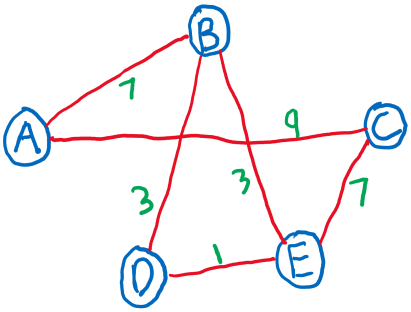
For ex., the output for the above graph would be `[["A","C"], ["B","A"], ["C","A"], ["D","B"], ["E","C"]]`.

Remember that the above graph would be represented as follows:

```
g = { "A" : [["B", 5], ["C", 9]],
      "B" : [["A", 5], ["D", 3], ["E", 4]],
      "C" : [["A", 9], ["E", 7]],
      "D" : [["B", 3], ["E", 1]],
      "E" : [["B", 4], ["C", 7], ["D", 1]] }
```

8. **Graphs CT:** Given the following mystery function, what would it return when given the graph below as input?

```
def f(g):
    result = {}
    for node in g:
        for neighbor in g[node]:
            if neighbor[1] not in result:
                result[neighbor[1]] = []
            result[neighbor[1]].append([node, neighbor[0]])
    return result
```



Remember that the above graph would be represented as follows:

```
g = { "A" : [["B", 7], ["C", 9]],
      "B" : [["A", 7], ["D", 3], ["E", 3]],
      "C" : [["A", 9], ["E", 7]],
      "D" : [["B", 3], ["E", 1]],
      "E" : [["B", 3], ["C", 7], ["D", 1]] }
```

9. **Tractability SA** - Consider a problem called 3COL: Given an undirected graph, can we color the vertices with 3 colors so that no two adjacent vertices share the same colors?
    a. Imagine we have an algorithm that solves the 3COL problem by trying all the possible combinations of colors for all the vertices. Is this algorithm tractable or intractable?
    b. Now we have an algorithm such that we go over each edge of the graph and check if both vertices connected by the edge have the same colors, if so we return False, else return True. Is this verifying process tractable or not?
    c. Based on your answers to parts i and ii, what complexity class is the 3COL problem in?

10. **Tractability SA -** Label the following Big-O complexities as either tractable or intractable
    a. O((5^n)log(n))
    b. O(n^247)
    c. O(n!)

Unit: Scaling up Computing

1. **Parallel Programming SA:** Answer the following True/False questions and explain the reasoning behind your answer.
   a. In multitasking, concurrent programs are run on a single CPU

   b. Multitasking and multiprocessing are synonyms meaning the same thing.

   c. Concurrency means that programs are running at the exact same time.

2. **Parallel Programming MC/SA:** Select which of the following are True:
   a. Deadlocks can be resolved by programs locking important resources that other programs need
   b. Pipelining is useful in scenarios where every sub-task is dependant on the previous sub-task and the task is performed only once
   c. The length of time of a pipelined process depends on the length of the longest task
   d. The mapper passes on its output directly to the reducer which then combines the output

3. **Parallel Programming SA:** What is the definition of deadlock? How can you usually fix deadlock?

4. **Parallel Programming FR:** A factory that makes action figures is set up so that every worker does three tasks: collecting the components (5min), gluing them together (5min), and painting the resulting figure (5min). Gluing requires some tidy-up time before switching to a new task (5min), and painting requires both set-up and tidy-up time when switching to a new task (5min each), so the worker's schedule over an hour currently looks like this:

	00:00	00:05	00:10	00:15	00:20	00:25	00:30	00:35	00:40	00:45	00:50	00:55
X	C	G	T	S	P	T	C	G	T	S	P	T
Y	C	G	T	S	P	T	C	G	T	S	P	T
Z	C	G	T	S	P	T	C	G	T	S	P	T

X, Y, and Z are workers. C is collect, G is glue, P is paint, T is tidy, and S is set-up

A) How many fully-set-up figures can 3 workers currently produce in 1 hour?

Use the concept of **pipelining** to adjust this schedule so that the workers can generate more figures in an hour. Fill in the table below using the same codes we used above. Your new schedule does not need to be ideal; it should just be better than the old one.

	00:00	00:05	00:10	00:15	00:20	00:25	00:30	00:35	00:40	00:45	00:50	00:55
X												
Y												
Z												

B) How many fully-set-up figures can 3 workers produce with your new schedule in 1 hour?

5. **Internet SA:** Which of the following best describes the features of a distributed system? **Select all that apply.**

    - ❏ Has many connected computers.
    - ❏ Has only one computer with many cores.
    - ❏ Can perform multitasking.
    - ❏ Can perform multiprocessing.
    - ❏ Can perform pipelining of tasks.
    - ❏ None of the above.

6. **Internet SA:** What is a packet? Name one thing that could go wrong when sending a packet, and how the Internet deals with it.

7. **Internet SA:** What is the relationship between an IP Address and a URL?

8. **Internet MC:** Match the following internet terms with their corresponding definitions:

Terms:	Definitions:
Packet	A computer that stores URL's and their associated IP addresses
DNS Server	A type of system that has failsafes and backups in place for when things go wrong
IP Address	Type of organization that connects a user's computer to the internet
Fault Tolerant	Information sent to an IP address
ISP	The set of numbers that uniquely identifies a computer hosting a website

9. **Internet SA:** For each of the following statements, select whether it is True or False.
   a. Websites split up data into a few large packets based on content before sending them through the internet.

   b. Packets all arrive at your browser in the correct order, to support buffering.

   c. Packets might all take different paths through routers to reach your computer.

10. **Security SA:** What does it mean to be fault tolerant? At a high level (no specific examples needed) how is the Internet structured to allow it to be fault tolerant?

11. **Security SA:** What are the two types of authentication we discussed in class? Briefly describe how they work and give an example of both types.

12. **Security SA:** Explain why good asymmetric encryption is secure. What might change that might make it NOT secure?

13. **Security SA:** Explain how asymmetric keys work differently from symmetric keys

14. **Security SA:** What is the difference between data privacy and data security? What do the two have in common?

15. **Security SA:** How can encryption help to stop a Man-in-the-Middle attack?

16. **Security SA:** Given the following scenarios, determine what type of security attack it is: DDOS or Man in the Middle.
    a. An attacker sets up a router and allows other people to connect to it. The attacker reads every packet that people send and records usernames and passwords.

    b. An attacker wants to prevent people from receiving website content, so they send many packets to the website's server to overwhelm it.

Unit: CS as a Tool

1. **Data Analysis CW: (Note that we are unlikely to ask you to write code with matplotlib on the final exam this semester, but we may ask you to read and interpret module documentation.)** Write a function that randomly samples 100 choices from a list that contains 4 flavors of ice cream (Vanilla, Chocolate, Mint, and Caramel), and then creates a bar chart of the number of times each flavor was sampled. For example, if each flavor is sampled 25 times, then the corresponding bar chart should have 4 bars (1 for each flavor) each with height 25. Use the
   random.choice() function from the random library, and use the plt.bar() function from the matplotlib library. The plt.bar() function takes two parameters: the first is a list of labels for the bars on the x-axis, and the second is the list of heights corresponding to each bar.

2. **Data Analysis SA:** Given the following types of data, name the best visualization method to display that data:
   a. Numerical × Ordinal
   b. Numerical
   c. Ordinal × Ordinal
   d. Categorical
   e. Numerical × Numerical × Categorical

3. **Simulation CW:** For this problem, you have two different circles. The entries in makeModel marked with a 1 correspond to the first circle and the entries marked with a 2 correspond to the second. Implement this simulation program by modifying `makeModel` and filling in `keyPressed` and `mousePressed`, assuming you have a 400x400 board:
    a. If the left half of the board is clicked, then only circle 1 is allowed to move
    b. If the right half is clicked, then only circle 2 is allowed to move
    c. If a half is clicked and it is the half of the circle currently allowed to move, then neither circle is allowed to move
    d. If the right or left arrow is pressed, then the circle that's currently allowed to move will move, if neither is allowed, then neither will move

```
def makeModel(data):
    data["cx1"] = 100
    data["cy1"] = 100
    data["cx2"] = 300
    data["cy2"] = 300
    data["size"] = 50
    data["color1"] = "cyan"
    data["color2"] = "pink"

def makeView(data, canvas):
    canvas.create_oval(data["cx1"] - data["size"],
                       data["cy1"] - data["size"],
                       data["cx1"] + data["size"],
                       data["cy1"] + data["size"],
                       fill=data["color1"])
    canvas.create_oval(data["cx2"] - data["size"],
                       data["cy2"] - data["size"],
                       data["cx2"] + data["size"],
                       data["cy2"] + data["size"],
                       fill=data["color2"])

def keyPressed(data, event):
    pass

def mousePressed(data, event):
    pass
```

4. **Simulation CW:** Given the setup below, write a function such that when you click on a random spot on the screen, a circle of size 50 is added to the board.
    a. Assume the board is 400x400
    b. The circle is added at the spot clicked and needs to have a randomly selected color(you need at least two colors on the board). (update mouseClicked)
    c. The circle will then start to float until it hits the top of the board (no specific bound, just be close to the top). (update runRules for this)

    We will use a dictionary implementation for the circles with relevant entries being displayed in makeview. These circles should be stored in a list in data.

```
def makeModel(data):
    # Set up your simulation components here
    data['circleDicts'] = [ ]

def makeView(data, canvas):
    # The simulation view is written here, using the Tkinter canvas
    for circle in data['circleDicts']:
        cx = circle['cx']
        cy = circle['cy']
        size = circle['size']
        color = circle['color']
        canvas.create_oval(cx-size, cy-size,
                           cx+size, cy+size, fill=color)

def runRules(data, call):
    pass

def mousePressed(data, event):
    pass
```

5. **Simulation CW:** Write a simulation function runTrial() that takes in 0 parameters and simulates rolling an eight-sided die 3 times. The function should return True if the first dice roll is EVEN, the second roll is ODD, and that the third roll is LESS THAN 5. Otherwise, it should return False.

    Ex: if Die 1 rolls a 4, Die 2 rolls a 7, and Die 3 rolls a 3 → returns True
    Ex2: If Die 1 rolls 2, Die 2 rolls 5, and Die 3 rolls a 8 → returns False

    Make sure to import the necessary modules!

6. **Simulation MC:** Which of these statements is/are false (can be more than one)?
    a. Computers can produce truly random numbers using functions from the random module
    b. Law of large numbers says that if the number of trials is around 100,000, the exact expected value will be reached.
    c. Testing a simulation with 10 runs of it will give a bad estimate of the expected value
    d. Monte Carlo methods are used to repeat simulations over and over again

7. **Machine Learning MC:** Categorize each of the following as either a classification, regression, or clustering problem:
    a. Based on a set of symptoms, determine what illness a patient has.
    b. Group a set of pictures into three groups, with similar pictures being in the same group
    c. Using the number of people who show up to a movie premiere to predict how much money the movie will make