Final Exam

Spring 2026

Name Answer Key

Net ID _____ (@uw.edu)

Academic Integrity: You may not use any resources on this exam except for your two-page (front and back of each) reference, writing instruments, your own brain, and the exam packet itself. This exam is otherwise 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 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
Asymptotic Analysis	7
Pre-Midterm DS	9
Hashing	12
Sorting	7
Graphs	13
Parallelism	10
Concurrency	9
Extra Credit	(+2)
Total	67

Section 1: Asymptotic Analysis

(4 pts)**Question 1: Better or Worse?**
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.)

```
public static void doStuff(BinaryMaxHeap<Integer> heap) {
    BinarySearchTree<Integer, Integer> tree = new BinarySearchTree();
    for (int i = 0; i < heap.size(); i++) {
        // for insert(), the first argument is key, the second argument is value
        tree.insert(heap.array[i].getPriority(), i);
    }
}
```

1. What is the **best-case** runtime of the program above?

   $\Theta($ n $)$

2. What is the **worst-case** runtime of the program above?

   $\Theta($ n^2 $)$

3. In one sentence describe what is the scenario for best case:

   When all items in the priority queue has the same priority

Now instead of a Binary Search Tree, suppose we use an AVL Tree to implement the program above:

```
public static void doStuff(BinaryMaxHeap<Integer> heap) {
    AVLTree<Integer, Integer>  tree = new AVLTree();
    for (int i = 0; i < heap.size(); i++) {
        tree.insert(heap.array[i].getPriority(), i);
    }
}
```

4. What is the **worst-case** runtime of the program above?

   $\Theta($ $n \log n$ $)$

(3 pts)**Question 2: Keeping up with the functions**

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. $f(n) = n!\,, g(n) = 2^n$
   - 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 $f(n) \notin \Theta(g(n))$

   C

2. $f(n) = (\log_3(n))^5, g(n) = (\log_5(n))^3$
   - 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 $f(n) \notin \Theta(g(n))$

   C

3. $f(n) = \log(n^{100}), g(n) = \log(2^{\log(n)})$
   - 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 $f(n) \notin \Theta(g(n))$

   A

Section 2: Pre-Midterm Data Structures

(3 pts)**Question 3: Vacuums, ATMs, and AI, oh my!**
For each scenario below, identify which of the ADT options provided is the most appropriate choice.
Options: Stack, Queue, Priority Queue, Ordered Dictionary

1. When an ATM is used, it photographs the user every 30 seconds. These photos, along with their time stamps, are stored in a data structure which will be used to look up pictures by range of time stamps (e.g. get all photos taken between 10:08 and 10:23). What ADT should this data structure implement?

   Ordered Dictionary

2. A robotic vacuum is cleaning a house. The robot begins at its docking station. It follows a path to clean its current room. If at any point it discovers a new room, it will pause the cleaning path of the current room and then begin a new path for the new room. After finishing with the new room it will return to the prior room, picking up where it left off. Which ADT should the vacuum use to store the rooms?

   Stack

3. I have an AI tool that movie producers use for generating music. Since computing time is limited, producers submit a prompt and a bid for how much they will pay for the result. We always process the prompt with the highest current bid. What ADT should be used for storing the bids?

   Priority Queue

(2 pts)**Question 4: Heap Arithmetic**
Assume that we have a Binary **Min Heap** with numbers $1, 2, 3, 4, 5 \cdots, 4096$ inserted and the binary min heap is stored using **0-indexed** array representation. (hint: $4096 = 2^{12}$)

1. What is the height of the heap? Recall that an empty heap has height -1.

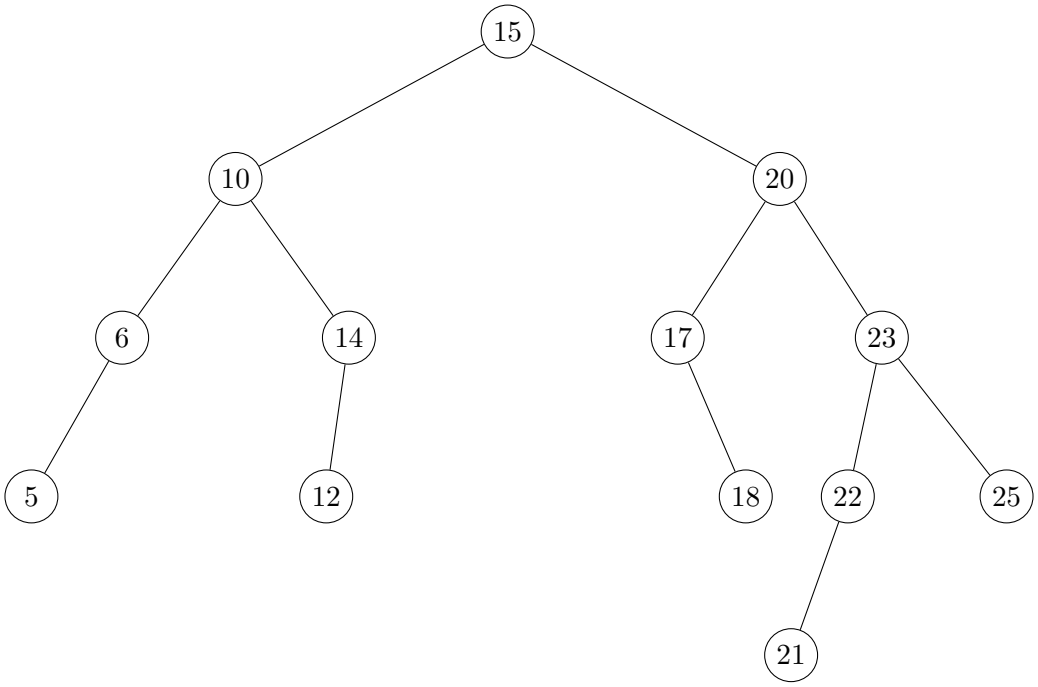
   12

2. What is the smallest value that might appear at index 4095?

   13

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

Answer the following questions about the AVL Tree below. Each question should be considered completely independently (i.e. "reset" to the image between questions).



1. Give an **integer** key which, when inserted into the given AVL tree, would cause a **double rotation** (either left-right or right-left)

   13

2. Give an **integer** key which, when inserted into the original AVL tree given above, would cause a **single right rotation**.

   4 or 11

Section 3: Hashing

(4 pts)**Question 6: Insert? clever title here**
Identify whether the following statement is True or False, then justify your choice in 1-2 sentences.

In any hash table, the worst case runtime for an `insert` would always be at least as large as `find` operation.

True or False?: True

Justification (1-2 sentences):

Need to check if the key is already present in the data structure (to determine if we are adding a new key-value pair or updating the value of a previously-added key).

(3 pts)**Question 7: Bad Hash Functions**
Recall that a good hash function must be **Consistent**, **Uniform**, **Effective**, and **Efficient**. Each subproblem below describes a symptom of a hash function missing one of these properties. Write which missing property best explains the behavior described.

1. No matter how many items I insert into my hash table, no key ever maps to index 7. The hash function is most likely not:

   Uniform

2. I inserted a key value pair into my hash table, then later did a find on that key. The find came back as unsuccessful even though I never called remove. The hash function is most likely not:

   Consistent

3. My key objects have a field called name and a field called title. I find that regardless of the title field, two keys with the same name always map to the same index. The hash function is most likely not:

   Effective

(5 pts)**Question 8: Quadratic Probing**
Insert 8,28,9,11,19,29 (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 quadratic probing for collision resolution.

Items that could not be inserted:

Index	Value
0	9
1	11
2	
3	19
4	
5	29
6	
7	
8	8
9	28

Section 4: Sorting

(4 pts)**Question 9: Sorting Suggestions Hotline**

Suppose you are responsible for a phone-in service where people call in, describe the items they need to sort, and you recommend an algorithm for them to use. For each scenario described below, indicate which sorting algorithm property the caller would most need, then identify the *fastest* sorting algorithm which possesses that property from the options provided.

1. I am developing a game leaderboard feature, the leaderboard shall be updated one score at a time as players finish matches, and only players who have finished will be displayed. I want the algorithm to be fast enough so that the user can see the information on leaderboard in real-time.

   Algorithms to choose: Insertion Sort, Merge Sort, Radix Sort, Quick Sort

   Properties to choose: Online, Adaptive, Stable, Non-Comparison Based

   Algorithm: Insertion Sort

   Property: Online

2. I am maintaining a unordered dictionary where US phone numbers are the keys. I have millions of items that I now need sorted in ascending order.

   Algorithms to choose: Insertion Sort, Heap Sort, Radix Sort, Quick Sort

   Properties to choose: Online, Adaptive, Stable, Non-Comparison Based

   Algorithm: Radix Sort

   Property: Non-Comparison Based

(3 pts)**Question 10: Radix Sort Optimization**

Consider that we wish to sort a list of $1024 = 2^{10}$ positive integers between 1 and $1,048,576 = 2^{20}$. Identify which of the two bases provided would result in the best running time for radix sort from the options provided. Justify your answer. **Options for base:** 2^{10} or 2^{20}

Best Base: 2^{10}

Justification:

running time: $(2^{10} + b)\log_b(2^{20})$
for $b = 2^{10}$: $(2^{10} + 2^{10})\log_{2^{10}}(2^{20}) = (2^{10} + 2^{10})2 = 2^{12}$
for $b = 2^{20}$: $(2^{10} + 2^{20})\log_{2^{20}}(2^{20}) = 2^{10} + 2^{20}$

Section 5: Graphs

(4 pts)**Question 11: Better than Dijkstra's?**
Recall that when Dijkstra's algorithm adds a node to the priority queue or updates a node's priority on the queue, it normally computes the priority in this way:

```
nextPriority = currPriority + weight(curr, next)
```

Consider if we instead used:

```
nextPriority = max(currPriority, weight(curr, next))
```

While all other parts of Dijkstra's algorithm remain unchanged.

1. Supposing we ran this modified Dijkstra's algorithm starting from node A, what option below best describes the output of this algorithm?

   ◯ The shortest paths from node A

   ◯ The longest paths from node A

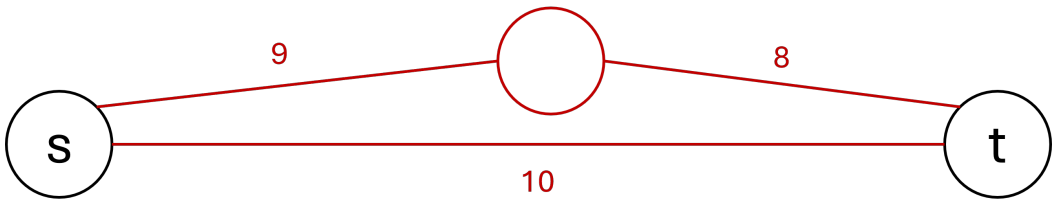
   ◉ The paths from node A that minimize the maximum edge weight

   ◯ The paths from node A that maximize the minimum edge weight

   ◯ A minimum spanning tree

   ◯ A maximum spanning tree

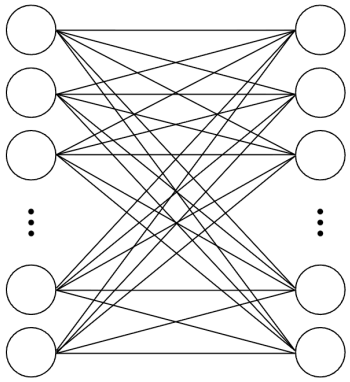
2. Add nodes, edges, and edge weights to the nodes labeled s and t below so that the modified Dijkstra's algorithm does not find the shortest path from s to t. Your graph may only use positive weight edges.



(4 pts)**Question 12: DFS and BFS**

Consider an undirected graph with $2n$ vertices divided into two groups: n vertices on the left side and n vertices on the right side, as shown in the picture below.

Every left-side vertex has an edge to every right-side vertex, and there are no other edges. Suppose a breadth-first search and depth-first search both start from a vertex on the left side.



1. What is the maximum number of vertices stored in the BFS queue at any point during execution? Give your answer in terms of n.
   (Hint: the node we start from does not impact the answer)

   2n - 2

2. Suppose DFS is implemented recursively. What is the maximum number of recursive calls on the call stack at any point during execution? Give your answer in terms of n.
   (Hint: the node we start from does not impact the answer)

   2n

3. What is the asymptotic running time of BFS on this graph?

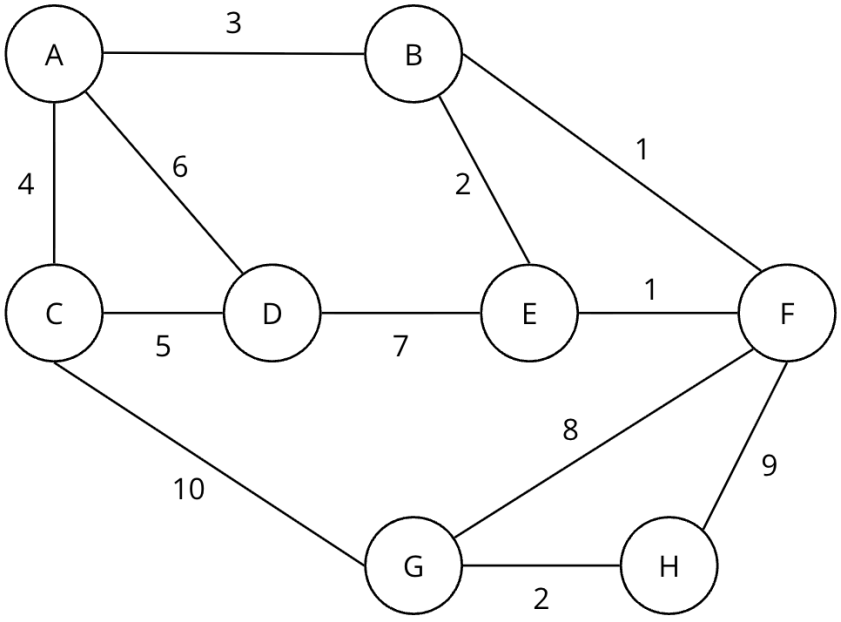
   ○ $\Theta(n)$

   ○ $\Theta(n \log n)$

   ● $\Theta(n^2)$

   ○ $\Theta(n^3)$

(5 pts)**Question 13: Building MSTs**
Suppose we have the following weighted, undirected graph (which also appears on the scratch work page if you wanted to draw on them):



1. Identify the first three edges added to the MST by **Kruskal's Algorithm**.

○ A–B	○ A–C	○ A–D
○ B–E	● B–F	○ C–D
○ C–G	○ D–E	● E–F
○ F–G	○ F–H	● G–H

2. Identify the first three edges added to the MST by **Prim's Algorithm** if we start from node A.

● A–B	○ A–C	○ A–D
○ B–E	● B–F	○ C–D
○ C–G	○ D–E	● E–F
○ F–G	○ F–H	○ G–H

3. What is the total cost of the whole MST? 24

Section 6: Parallelism

(4 pts)**Question 14: Forkjoin**

Below we have a binary search tree class called `Tree`. This class includes a method called `countGreaterThan` (implemented sequentially) which returns the number of nodes in the tree whose `data` field is greater than `x`.

On the next page we have an incomplete parallel implementation of `countGreaterThan` using ForkJoin, for this question you will finish that parallel implementation.

```
public class Tree {
    public Node root;

    public class Node {
        public int data;
        public Node left;
        public Node right;
    }

    public int countGreaterThan(int x) {
        return count(root, x);
    }

    public int count(Node curr, int x) {
        if (curr == null) {
            return 0;
        }

        int leftCount = count(curr.left, x);
        int rightCount = count(curr.right, x);

        if (curr.data > x) {
            return 1 + leftCount + rightCount;
        } else {
            return leftCount + rightCount;
        }
    }
}
```

Below is an incomplete parallel implementation of `countGreaterThan` using ForkJoin. We have provided:

– A `Tree` class that has the `countGreaterThan` method.

– Fields for a `CountTask` class.

– The constructor for the `CountTask` class.

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

And the code is missing:

– The body of the `countGreaterThan` method, which should create an instance of `CountTask` and call the `invoke` method of `ForkJoinPool`.

– The class that `CountTask` should extend.

– The parallelized portion of the `compute` method.

```
1  import java.util.concurrent.*;
2  public class Tree {
3      private Node root;
4      public static final ForkJoinPool POOL = new ForkJoinPool();
5
6      public class Node {
7          public int data;
8          public Node left;
9          public Node right;
10     }
11
12     public int countGreaterThan(int x) {
13         // Part 1 answer will go here
14
15     }
16 }
17
18 public class CountTask extends ??? { // Part 2 will replace the ???
19     private Node curr;
20     private int x;
21
22     public CountTask(Node curr, int x) {
23         this.curr = curr;
24         this.x = x;
25     }
26
27     public Integer compute() {
28         if (curr == null) {
29             return 0;
30         }
31         // Part 3 answer will go here
32
33     }
34 }
```

Finish the parallel implementation by completing the boxes below. Its behavior should match the sequential implementation.

1. Implement the body of `countGreaterThan` in the box provided. Your code will start at line 13.

```
return POOL.invoke(new CountTask(root, x));
```

2. Finish the declaration of `CountTask` on line 18 by filling in what class it should extend.

```
RecursiveTask<Integer>
```

3. Finish the `compute` method. This code will start at line 31.

```
CountTask left = new CountTask(curr.left, x);
CountTask right = new CountTask(curr.right, x);

left.fork();
int rightCount = right.compute();
int leftCount = left.join();

int currCount = curr.data > x ? 1 : 0;

return currCount + leftCount + rightCount;
```


(4 pts)**Question 15: Work and Span**

Suppose the binary tree from the previous question contains n nodes and is perfectly balanced.

Assume that each node requires $O(1)$ work, and that task creation/join costs are also $O(1)$.

Give tight asymptotic bounds for:

1. Total work, $T_1(n) = \Theta($ n $)$
2. Span, $T_\infty(n) = \Theta($ $\log n$ $)$

(2 pts)**Question 16: Parallel Pack/Filter**

Below we have displayed just one portion each of the input array, map result array, and prefix sum array used in the parallel pack/filter algorithm. Answer the questions below relating to the output array.

input array:

...	9	2	8	12	5	20	13	40	1	0	14	18	22	34	23	7	6	81	19	42	...
	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	

map result array:

...	0	1	1	0	1	0	1	1	1	0	1	0	0	1	0	1	1	0	1	0	...
	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	

prefix sum array:

...	25	26	27	27	28	28	29	30	31	31	32	32	32	33	33	34	35	35	36	36	...
	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	

1. What value will we find at the index 32 of the output array?

   34

2. What value will we find at the index 33 of the output array?

   7

Section 7: Concurrency

(5 pts)**Question 17: Race Conditions**

Consider if we had multiple threads sharing a single instance of the Course class provided below. The expected behavior is that the number of students enrolled in a course should never exceed the provided capacity of the course. Assume there are no concurrency issues within the `ArrayList` class.

```
1   public class Course{
2       private String name;
3       private ArrayList<Student> roster;
4       private int capacity;
5       private int size;
6
7       public Course(String name, int capacity){
8           this.roster = new ArrayList<>();
9           this.size = 0;
10          this.capacity = capacity;
11      }
12
13      public void enroll(Student student){
14          if(size < capacity){
15              synchronized(roster){
16                  roster.add(Student);
17              }
18              size = size + 1;
19          }
20      }
21  }
```

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

1. Describe an interleaving in which the number of students in the roster exceeds the course's capacity (i.e. `roster.size() > capacity`).

   Thread 1 checks the if statement, calls roster.add(), then reads the size field. Thread 2 checks the if statement, calls roster.add, then calls size = size + 1. Finally thread 2 calls size=size+1 using the already-accessed size field.

2. Consider if we change line 18 to become `size = roster.size();`? This makes the code:

```
public void enroll(Student student){
    if(size < capacity){
        synchronized(roster){
            roster.add(Student);
        }
        size = roster.size();
    }
}
```

   Is it possible to have `roster.size() > capacity`? Indicate "yes" or "no".

   yes

3. Consider if we move all of the method's body to be inside the synchronized block, making the code:

```
public void enroll(Student student){
    synchronized(roster){
        if(size < capacity){
            roster.add(Student);
            size = size + 1;
        }
    }
}
```

   Is it possible to have `roster.size() > capacity`? Indicate "yes" or "no".

   no

4. Consider if we change line 18 to become `size = roster.size();` and then move the if statement to be inside of the synchronized block (and then update the size outside of the if statement)? This makes the code:

```
public void enroll(Student student){
    synchronized(roster){
        if(size < capacity){
            roster.add(Student);
        }
    }
    size = roster.size();
}
```

   Is it possible to have `roster.size() > capacity`? Indicate "yes" or "no".

   yes

(4 pts)**Question 18: Deadlock**

Consider that we have the `Registrar` class defined below to keep track of course enrollments and student grades. The `Registrar` class references the `Course` and `Student` classes additionally provided.

This `Registrar` class has the potential for deadlock. Questions relating to this appear on the next page.

The bodies of the `Course` and `Student` methods as well as constructors for the classes are not necessary for this question, and so they have been omitted.

```
1  public class Registrar{
2      public void enrollSchedule(Student student, List<Courses> schedule){
3          for(Course course : schedule){
4              synchronized(course){
5                  if(course.enroll(student)){
6                      student.addCourse(course);
7                  }
8              }
9          }
10     }
11     public void buildTranscript(Student student){
12         synchronized(student){
13             for(Course c : student.schedule){
14                 System.out.println(c.getGrade(student));
15             }
16         }
17     }
18 }
19
20 public class Course{
21     public synchronized double getGrade(Student s){...}
22     public synchronized boolean enroll(Student s){...}
23 }
24
25 public class Student{
26     public synchronized addCourse(Course c){...}
27 }
```

1. Two threads invoking methods from the `Registrar` class has a potential for deadlock. Explain how deadlock might occur.

   Suppose we have thread 1 calling enrollSchedule and thread 2 calling build transcript. Consider there is a student-course pair where the course is on the schedule and the student is in the roster for the course already. In this case thread 1 might obtain the lock for that course on line 7, then the lock for the student on line 9. Thread 2 would then obtain the lock for the student on line 15, then the lock for the course on line 17. We now have 2 locks held simultaneously in opposite orders, resulting in deadlock.

2. Suppose we added a statement to synchronize on the student at the start of the method, resulting in the following code:

```
public void enrollSchedule(Student student, List<Courses> schedule){
    synchronized(student){
        for(Course course : schedule){
            synchronized(course){
                if(course.enroll(student)){
                    student.addCourse(course);
                }
            }
        }
    }
}
```

   Does this new code have the potential for deadlock? Answer "yes" or "no".

   no

3. Suppose we instead made all methods in `Registrar` synchronized, so that they had the following signatures:

```
public synchronized void enrollSchedule(Student student, List<Courses> schedule){...}
public synchronized void buildTranscript(Student student){...}
```

   Does this new code have the potential for deadlock? Answer "yes" or "no".

   no

Section 8: Extra Credit

(+1 pts)**Question Extra 1: Nathan Brunelle**
Which of these would you be most surprised to hear is our professor?


(a)


(b)


(c)


(d)


(e)

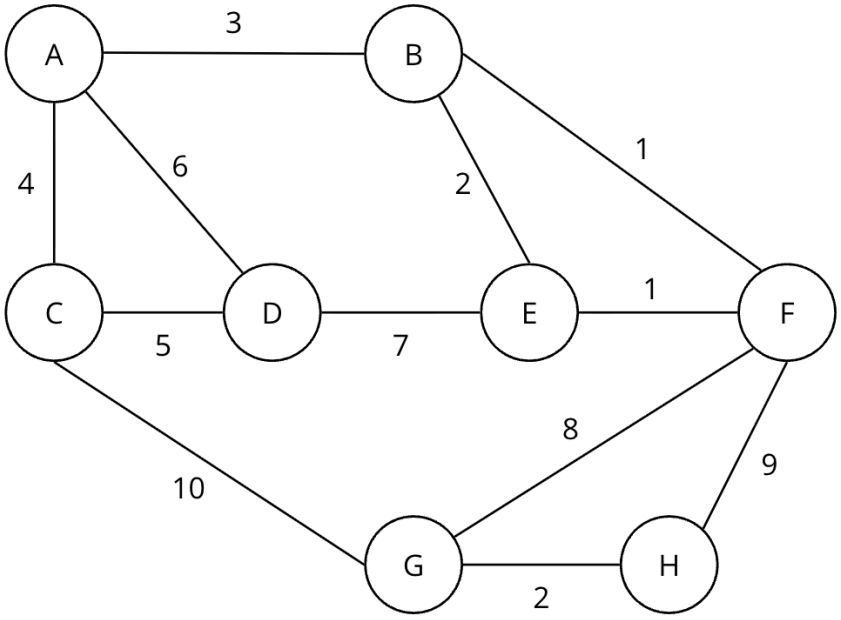
Explain your reasoning:

The one on the left in c, because my professor is not a dog.

Scratch Work

Nothing written on this page will be graded.

Graph used for question 13.



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

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

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

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

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

