CSE 332 Autumn 2025 Final Exam

Your
Seat Number: _____

Name: ____**Sample Solution** _____

UW NetID: _____(@uw.edu)

Instructions:

- The allotted time is 1 hour and 50 minutes.
- Please do not turn the page until the staff says to do so.
- This is a closed-book and closed-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- When provided, write your answers in the box or on the line provided.
- Unless otherwise noted, every time we ask for an O, Ω, or Θ bound, it must be simplified and tight.
- For answers that involve bubbling in a ◯ or ☐, make sure to fill in the shape completely.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.
- A formula sheet has been included at the end of the exam.

Advice:

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Total: 92 points

Q1: Short-answer questions (20 pts)

- For questions asking you about runtime, give a simplified, tight Big-O bound. This means that, for example, $O(5n^2 + 7n + 3)$ (not simplified) or $O(2^{n!})$ (not tight enough) are unlikely to get points. Unless otherwise specified, all logs are base 2.
- Unless otherwise specified, assume the optimal implementation of a data structure or version of algorithm discussed in lecture is used.

We will only grade what is in the provided answer box.

a. **True** or **False**: The **worst case** runtime of finding the **maximum** value in an **AVL tree** containing N elements is $\Omega(N)$.

○ True ● False

b. Give a **worst case** recurrence for **find** in a **binary search tree** containing n elements. Use variables appropriately for constants (e.g. c1, c2, etc.).

$T(1) = c_0$

$T(n) =$ T(n - 1) + c1

c. Give the **worst case** runtime for **push** on a **stack** implemented using singly-linked list nodes, containing N elements.

O(1)

d. Give the **worst case** runtime for creating a **binary max heap** from the values in an **AVL tree** containing N elements.

O(N)

e. What is the **minimum** number of nodes in a **binary min heap** with height 6? (Remember: A single node is a tree of height 0.) **Note: we are looking for the exact number here.** For this question partial credit will <u>not</u> be given for formulas or anything other than the actual number - check your work!

64

Q1: (continued)

f. Give the exact number for the minimum number of edges in a connected undirected graph containing **9 vertices**. **Note: we are looking for the exact number here.** For this question partial credit will not be given for formulas or anything other than the actual number - check your work!

8

g. The **span** of the **parallel pack** algorithm run on an array of size N, as described in lecture, is:

O(log N)

h. Give the runtime of **Merge sort** on an array of size N if the array happens to be already sorted.

O(N log N)

i. What is the **worst case** runtime of a **breadth first search** of a directed graph containing V vertices and E edges. Assume an adjacency list representation (as described in lecture) of the graph is used and give your answer in terms of V and/or E.

O(V + E)

j. For a program where at most ¾ of it can be parallelized, what is the **maximum speedup** you would expect to get with 9 processors? **Note: we are looking for the exact number here.** For this question partial credit will not be given for formulas or anything other than the actual number - check your work!

3

Q2: Hashing (8 pts)

a) [3 pts] **Double Hashing Hashtable**. Insert **71, 3, 6, 42, 13, 2** into the table below (TableSize = **10)**. You should use the primary hash function: **h(k) = k % 10** and a secondary hash function: **g(k) = (k % 3) + 2**.
If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values. Assume no re-sizing occurs during these insertions.

If any values cannot be inserted, write them here:_____

0	2
1	71
2	42
3	3
4	
5	
6	6
7	
8	
9	13

b) [1 pt] After attempting to insert the elements above, what is the load factor for the table in part a)?

6/10

c) [2 pt] What is the **worst case** runtime of a **find** operation in a **separate chaining** hash table that does not allow deletions. The table contains N elements,Tablesize is N^3 and each bucket points to an **AVL tree**?

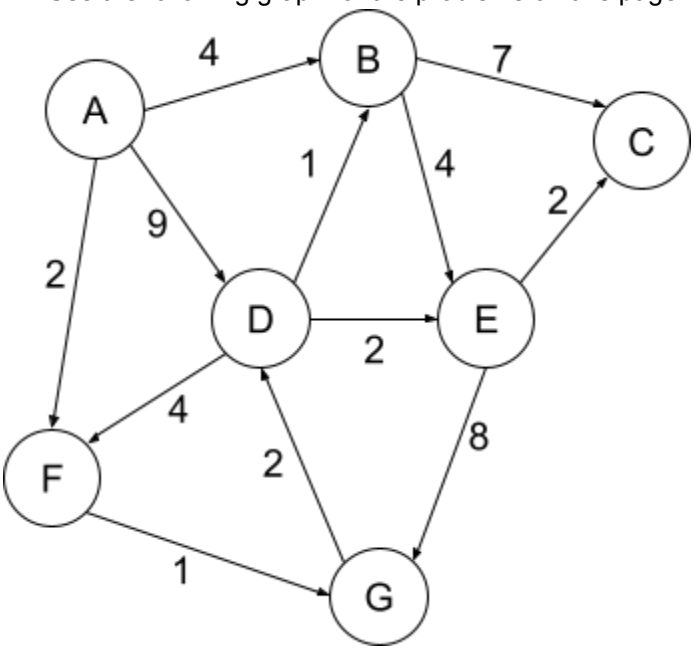
O(log N)

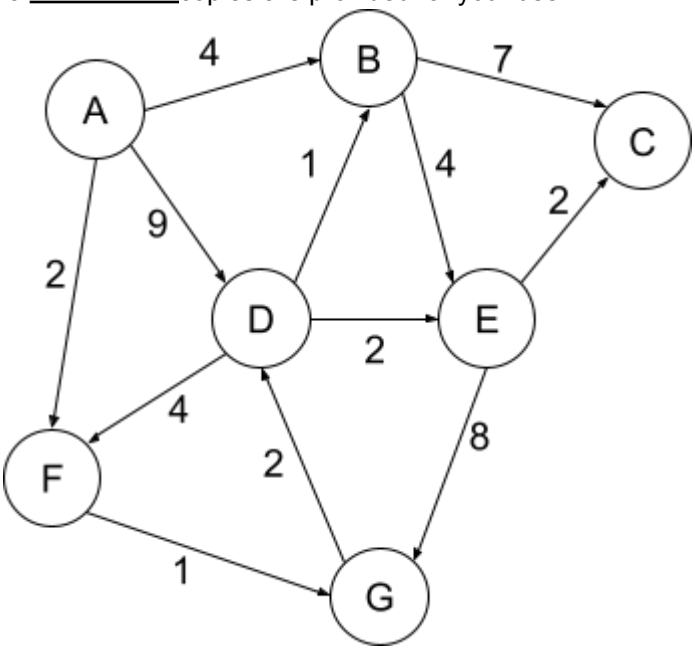
d) [2 pt] What is the **worst case** runtime of a **find** operation in a **linear probing** hash table that does not allow deletions. The table contains N elements and Tablesize is N^3?

O(N)

Q3: Dijkstra's Algorithm (6 pts)

Use the following graph for the problems on this page. Two **IDENTICAL** copies are provided for your use:





a) [4 pts] Step through Dijkstra's Algorithm to calculate the **single source shortest path from A** to every other vertex (in terms of cost, not number of edges). Break ties by choosing the lexicographically smallest letter first; ex. if B and C were tied, you would explore B first. ***Note that the next question (part b) asks you to recall what order vertices were declared known*.** Make sure the final distance and predecessor are clear in the table below.

Vertex	Known	Distance	Predecessor
A	T	0	------
B	T	4	A
C	T	~~11,~~ 9	~~B~~, E
D	T	~~9~~, 5	~~A~~, G
E	T	~~8~~, 7	~~B~~, D
F	T	2	A
G	T	3	F

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

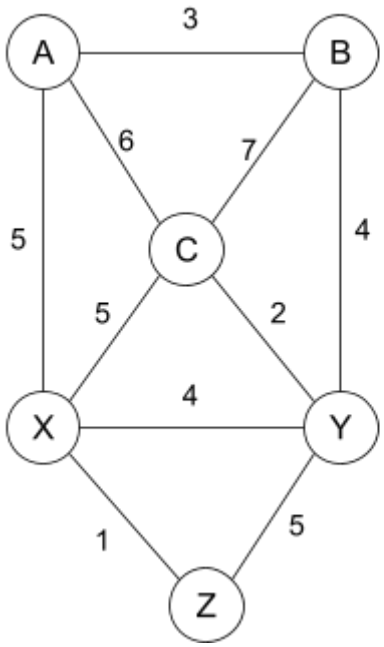
__A___, __F___, __G___, __B___, __D___, __E__, ___C__

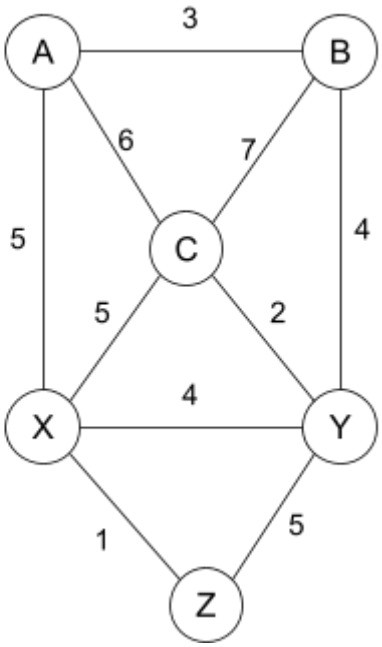
c) [1 pt] List the **shortest path** from A to C. (Give the actual path **NOT** the cost.)

A, F, G, D, E, C

Q4: More Graphs (7 pts)

Use the following graph for the problems on this page. Two **IDENTICAL** copies are provided for your use:





a) [2 pts] When performing Kruskal's algorithm on the graph,
immediately after the edge between C and Y is processed:

i) How many disconnected trees exist? 4

ii) **How many vertices are in the tree containing the largest number of vertices**? 2

b) [2 pt] What is the **total cost** of a minimum spanning tree found by Kruskal's algorithm in the graph above? 14

c) [1 pt] **True** or **False**: Prim's algorithm **starting at vertex A** would find the same minimum spanning tree as Kruskal's algorithm. Ties in selection of edges are broken by choosing the edge containing the vertex that comes first alphabetically.

● True ○ False

d) [2 pts] ASSUMING the edges above are **unweighted**, give a valid **breadth first search** of this graph, **starting at vertex Z**, using the algorithm described in lecture (e.g. give the order that the nodes are marked as "finished"). **When adding elements to the data structure, you should break ties by choosing the lexicographically smallest letter first**; ex. if A and B were tied, you would add A to the data structure first. You only need to show the final breadth first search.

_Z____, _X____, __Y___, _A____, __C___, __B___

Q5: Sorting (10 pts)

** For Q5, all sorting algorithms **are the versions described in lecture.** **

a) [2 pt] Using median-of-three pivot selection as described in lecture, the **worst case** runtime for Quick Sort on an array of size N is:

O(N^2)

b) [2 pt] True or False: **Insertion** Sort is a **stable** sort.

● True ○ False

c) [2 pt] True or False: **Merge** Sort is an **in-place** sort.

○ True ● False

d) [2 pt] True or False: If we use a stable sort as the intermediate sort in **Radix** Sort for all passes except the last one, we can use an unstable sort for the last pass and still get correct results.

○ True ● False

e) [2 pts] Give the *recurrence* for Quick Sort on an array of size n in the **best case**. Use variables appropriately for constants (e.g. c_1, c_2, etc.).

$T(1) = c_0$

$T(n) =$ $2\,T(n/2) + c1*n + c2$

Q6: ForkJoin (14 pts)

In Java using the ForkJoin Framework, write code to solve the following problem:

- **Input**: A non-empty array of integers.
- **Output**: Returns the range (max - min + 1) of the elements in the array.

For example: Input array: {20, 3, 5, 10, 11, 100} returns 100 - 3 + 1 = 98.

Input array: {99, 100, 100, -3, 99} returns 100 - (-3) + 1 = 104.

- **Do **not** employ a sequential cut-off: **the base case should process one element**.**
  - i.e. you do not **need** to employ a sequential method and you can assume that the code will never process more than one element
- Give a class definition, `FindRangeTask`, along with any other code or classes needed.
- Fill in all of the **blanks** in the function `findRange` **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.

Hint: Feel free to use the functions `Math.min` and `Math.max` in your code..

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

public class Pair { // You should use this class
    public int min, max;
    public Pair (int min, int max) {
        this.min = min;
        this.max = max;
    }
}

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

    // Returns the range of the elements in the array.
    public static int findRange(int[] input) {

      __Pair p__ = pool.invoke(new FindRangeTask(___input, 0, input.length____));

      return _______ p.max - p.min + 1_____________________;
    }
}
```

Please fill in the three _____ in the function above and write your class on the next page.

****Don't forget to fill in the three blank lines above!!!!

Write your class here:

```
public class FindRangeTask extends  __RecursiveTask<Pair>_____  {
    // Fields go here
    private int[] array;
    private int lo;
    private int hi;

    public FindRangeTask(_int[] array, int lo, int hi_______) {
      this.lo = lo;
      this.hi = hi;
      this.array = array;
    }

    public __Pair__compute() {
      if (hi - lo == 1) {
          return new Pair(array[lo], array[lo]);
      } else {
          int mid = lo + (hi - lo) / 2;
          FindRangeTask left = new FindRangeTask(array, lo, mid);
          FindRangeTask right = new FindRangeTask(array, mid, hi);

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

          int min, max;
          min = Math.min(leftResult.min, rightResult.min);
          max = Math.max(leftResult.max, rightResult.max);

          /*
          if (leftResult.min < rightResult.min)
            min = leftResult.min;
          else
            min = rightResult.min;

          if (leftResult.max > rightResult.max)
            max = leftResult.max;
          else
            max = rightResult.max;
          */
          return new Pair(min, max);
    }
}
```

****Don't forget to fill in the <u>three</u> blank lines on THIS PAGE and the PREVIOUS PAGE!!!!*****

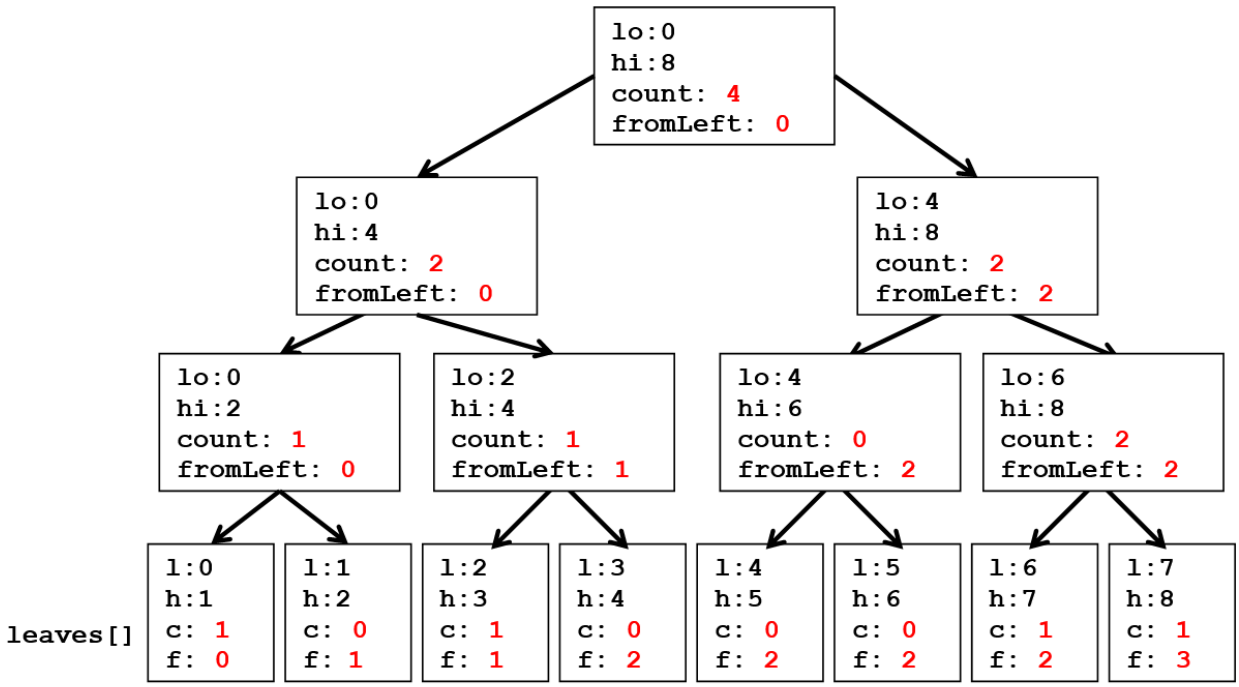
Q7: Parallel Prefix (9 pts)

Given the following array of integers as input, perform the parallel prefix algorithm to fill the output array with the **count of the values greater than 7 contained in all of the cells to the left** in the `input` array (including the value contained in that cell). Do not use a sequential cutoff.

For example, for `input = {3, 7, 10, 13, 8, 7, -6, 12}`,
`output` should be: `{0, 0, 1, 2, 3, 3, 3, 4}`.

a) [5 pts] Fill in the values for `count`, `fromleft`, and the `output` array in the picture below given the following values for `input`. The `input` array has already been filled out for you. Note that problems b-e, on the next page, ask you to give the formulas you used in your calculation

```
int[] input = {8, 6, 30, 7, 3, -5, 8, 22}
```



Index	0	1	2	3	4	5	6	7
input	8	6	30	7	3	-5	8	22
output	1	1	2	2	2	2	3	4

Q7 (continued)

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 on the previous page.

b) [1 pt] Give pseudocode for how you assigned a value to **leaves[i].count**

```
if (input[i] > 7)
      leaves[i].count = 1;
else
      leaves[i].count = 0;
```

c) [1 pt] Give code for assigning **p.left.fromLeft.**

```
p.left.fromLeft = p.fromLeft;
```

d) [1 pt] Give code for assigning **p.right.fromLeft.**

```
p.right.fromLeft = p.fromLeft + p.left.count;
```

e) [1 pt] How is **output[i]** computed? Give <u>exact code</u> assuming **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 on the previous page.

```
output[i] = leaves[i].count + leaves[i].fromLeft;
```

Q8: Concurrency (12 pts)

The **FastIntegerArray** class manages an array. Multiple threads might be accessing the same **FastIntegerArray** object. Code for the entire class shown below.

```
public class FastIntegerArray {
      private int[] array;
      private int size = 0;
      private ReentrantLock lock = new ReentrantLock();

      public FastIntegerArray(int maxSize){
            array = new int[maxSize];
      }

      public boolean isFull(){
            lock.acquire();
            boolean isfull = (size == array.length);
            lock.release();
            return isfull;
      }

      public boolean add(int x){
            if (this.isFull()){
                  return false;
            }
            lock.acquire();
            array[size] = x;
            size++;
            lock.release();
            return true;
      }

      public int get(int index){
            lock.acquire();
            if (index >= this.size){
                  throw new IndexOutOfBoundsException();
            }
            int returnVal = array[index];
            lock.release();
            return returnVal;
      }
}
```

a) [3 pts] Does the **`FastIntegerArray`** class have (bubble in all that apply):

■ a race condition ■ potential for deadlock ☐ a data race ☐ none of these

Give an explanation for each box you checked above (1-2 sentences each). Refer to line numbers in your explanation. Be specific!

Race Condition: due to a bad interleaving, not due to a data race. Two threads can pass the **`this.isFull()`** check at line 20 in **`add`**, when **`array`** is one away from being full. The first thread to add inserts **`x`** at **`array.length-1`**. Then the next thread tries to insert at **`array.length`** causing an error.

Deadlock: **`get`** doesn't release the lock at line 31 after throwing an error. This causes deadlock since the lock is never released and other threads will wait forever.

b) [3 pts] We now add this method to the **`FastIntegerArray`** class:

```
public int size(){
      return this.size;
}
```

Does adding this method to the **`FastIntegerArray`** class **cause any new** (bubble in all that apply):

■ a race condition ☐ potential for deadlock ■ a data race ☐ none of these

If there are any new problems, give an explanation for each box you checked above (1-2 sentences each). Refer to line numbers in your explanation. Be specific!

Data Race: A read of **`this.size`** on line 39 in **`size()`** could be happening at the same time another thread could be writing **`this.size`** on line 23 in **`add`**.

Race Condition: A Data Race is a Race Condition.

c) [6 pts] **On the next page** we have copied all the code from parts (a) and (b) with some extra space and **have removed the original lock**. Modify **the code** 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();
```

******Please modify the code on the next page! Do not modify the code on this or the previous page!****

Q8 Continued: Please Modify the code on THIS page.

```
public class FastIntegerArray {
	private int[] array;
	private int size = 0;
	// Declare any locks needed here
	private ReentrantLock lock = new ReentrantLock();

	public FastIntegerArray(int maxSize){
		array = new int[maxSize];
	}

	public boolean isFull(){
		lock.acquire();
		boolean isfull = (size == array.length);
		lock.release();
		return isfull;
	}

	public boolean add(int x){
		lock.acquire();
		if (this.isFull()){
			lock.release();
			return false;
		}
		array[size] = x;
		size++;
		lock.release();
		return true;
	}
	public int get(int index){
		lock.acquire();
		if (index >= this.size){
			lock.release();
			throw new IndexOutOfBoundsException();
		}
		int returnVal = array[index];
		lock.release();
		// Having lock.release() above the array access is also correct,
		// since size only increases and we never write a location
		// in the array more than once.
		return returnVal;
	}
	public int size(){
		lock.acquire();
		int size = this.size;
		lock.release();
		return size;
		return this.size;
	}
}
```

Q9: Pre-Midterm Medley (6 pts):

Describe the **worst-case** running time for the following pseudocode functions in Big-O notation in terms of the variable n. Your answer **MUST** be tight and simplified. **You do not have to show work or justify your answers for this problem.**

a)

```
int snow(int n, int inches) {
      for (int i = 0; i < n * n; i++) {
            if (i % 5 == 0) {
                  for (int j = 0; j < i; j++) {
                        inches++;
                  }
            }
      }
      return inches;
 }
```

O(n^4)

b)

```
int sled(int n, int feet) {
      if (n < 5) {
            return feet;
      } else {
            for (int i = 0; i < n; i++) {
                  feet++;
            }
      }
      return sled(n-2, feet);
 }
```

O(n^2)

This is a blank page! Enjoy!

Summations

1. $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ for $|x| < 1$
2. $\sum_{i=1}^{n} cf(i) = c\sum_{i=1}^{n} f(i)$
3. $\sum_{i=0}^{n-1} 1 = \sum_{i=1}^{n} 1 = n$
4. $\sum_{i=0}^{n} i = 0 + \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
5. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
6. $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
7. $\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$
8. $\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$

Logs:

1. $a^{\log_b(c)} = c^{\log_b(a)}$
2. $\log_b(a) = \dfrac{\log_d(a)}{\log_d(b)}$
3. $\log_b(b) = 1$
4. $\log_b(1) = 0$
5. $b^{\log_b(n)} = n$
6. $\log_b(n \cdot m) = \log_b(n) + \log_b(m)$
7. $\log_b(\frac{n}{m}) = \log_b(n) - \log_b(m)$
8. $\log_b(n^k) = k \cdot \log_b(n)$

This is another blank page! Enjoy!