

PRINT Your Name: _____

PRINT Your Student ID: _____

PRINT Student name to your left: _____

PRINT Student name to your right: _____

You have 110 minutes. There are 8 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	8	Total
Points:	12	8	9	19	18	14	20	0	100

For questions with **circular bubbles**, select only one choice (there is only one correct answer).

- ☐ Unselected option (Completely unfilled)
- ☒ Don't do this (it will be graded as incorrect)
- ☒ Only one selected option (completely filled)

For questions with **square boxes**, you may select one or more choices (select all that apply).

- ☐ You can select
- ☐ multiple squares
- ☒ Don't do this (it will be graded as incorrect)

- Anything you write outside the answer boxes or you ~~cross-out~~ will not be graded. If you write multiple answers or your answer is ambiguous, we will grade the **worst** interpretation.
- Unless otherwise specified, all data structures and algorithms behave according to their implementation in lecture or lab, with no additional optimizations, and all relevant libraries have been imported.
- If an implementation detail (e.g. tiebreaking scheme, linked list topology) is relevant, it will be explicitly noted in the question.
- You may write at most one statement per blank and you may not use more blanks than provided.
- Your answer will be reformatted according to the 61B/L style guidelines. For example, any method, constructor, or if-statement requires at least three lines for the purposes of determining line count.
- You may not use ternary operators, lambdas, streams, or multiple assignment.

Read the honor code below and sign your name.

By signing below, I affirm that all work on this exam is my own work. I have not referenced any disallowed materials, nor collaborated with anyone else on this exam. I understand that if I cheat on the exam, I may face the penalty of an "F" grade and a referral to the Center for Student Conduct.
--

SIGN your name: _____

Q1 Amy's Ponies

(12 points)

Consider the code below.

```
public interface Magical {
    default void useMagic() {
        System.out.println("ZAP!");
    }
}

public class Horse {
    public static String species = "equus";
    private int weight;

    public Horse() {
        this.weight = 900;
        System.out.println("Horse");
    }

    public Horse(int weight) {
        this.weight = weight;
        System.out.println("Weight: " + weight);
    }

    public void neigh() {
        System.out.println("Neigh!");
    }
}

public class Unicorn extends Horse implements Magical {
    private int magicLevel;

    public Unicorn(int magicLevel, int weight) {
        super(weight);
        this.magicLevel = magicLevel;
    }

    @Override
    public void neigh() {
        System.out.println("Friendship!");
    }

    @Override
    public void useMagic() {
        System.out.println("MAGIC!");
    }
}
```

(Question 1 continued...)

Q1.1 (12 points)

Write what each line would output in the box to the right of that line.

If the line would result in a compiler error, leave the box blank and bubble “CE”. If the line would result in a runtime error, leave the box blank and bubble “RE”. In either error case, continue as if that line was never run.

If no output is printed, and the line would not error, write the word “nothing” (without the quotes).

The first few lines have been given for you.

```
System.out.println("Welcome!");
```

☐ CE ☐ RE

Welcome!

```
String to = "the";
```

☐ CE ☐ RE

nothing

```
int exam = "!";
```

☒ CE ☐ RE

```
Horse appleJack = new Horse();
```

☐ CE ☐ RE

```
Magical pinkiePie = new Horse();
```

☐ CE ☐ RE

```
Unicorn twilight = new Unicorn(10, 50);
```

☐ CE ☐ RE

```
Magical sparkle = twilight;
```

☐ CE ☐ RE

```
Unicorn princessCelestia = new Horse();
```

☐ CE ☐ RE

```
Unicorn rarity = (Unicorn) appleJack;
```

☐ CE ☐ RE

```
twilight.neigh();
```

☐ CE ☐ RE

```
twilight.useMagic();
```

☐ CE ☐ RE

Q2 Addicted Drinkers of Tea (ADTs)

(8 points)

Noah, Wilson, and Karen all want to buy boba for every **Student** in the world. For each of the below subquestions, choose the *best* abstract data type (ADT) to use. Each subpart is independent from the other subparts.

Your options for ADTs are **List**, **Set**, **Map**, **Queue**, or **Stack**.

Q2.1 (2 points) Noah would like to know how far different buildings (represented by the **Building** class) are from CS61BobaShop. For example, Evans Hall is 61.8 kilometers from CS61BobaShop.

Write your chosen ADT, including the generic type(s):

Example answers: **List<Building>** or **Map<Integer, String>**

Q2.2 (2 points) Students (represented by the **Student** class) who order last receive their boba first (slightly unfair, yes). Wilson wants to keep track of the next student to receive their boba. Students are tracked immediately after ordering.

Write your chosen ADT, including the generic type(s):

Q2.3 (2 points) Karen asks students (represented by the **Student** class) to go into a tree formation, where every student represents a node. When distributing boba, Karen gives them to students in increasing order of distance from the root node (breaking ties arbitrarily). As she distributes boba, Karen wants to know which student is the next to receive boba.

Write your chosen ADT, including the generic type(s):

Q2.4 (2 points) Every time a student (represented by the **Student** class) mentions a boba type (represented by the **Boba** class), Michelle wants to know whether or not that boba type was mentioned before by *any* student.

Write your chosen ADT, including the generic type(s):

Q3 Would You Like to Get Foo'd

(9 points)

Q3.1 (9 points) Consider the following code:

```
class Foo {  
    private int x;  
    public Foo(int x) {  
        this.x = x;  
    }  
}
```

Codey is considering implementations for `boolean equals(Object obj) { ... }` in `Foo`.

We instantiate three `Foo` objects `foo1`, `foo2`, `foo3`, each with an arbitrary (not necessarily unique) value of `x`.

Then, we create a new `Set` according to some `Set` implementation, and insert `foo1`, `foo2`, and `foo3`, in that order, into this newly created `Set`.

Select whether each `equals` implementation always, sometimes, or never results in the respective final resulting `Set` after inserting all of `foo1`, `foo2`, `foo3`, in that order.

Note: Assume nonempty `Sets` call `equals` when checking for duplicates.

equals implementation	Final result after inserting all of <code>foo1</code> , <code>foo2</code> , <code>foo3</code> into an empty <code>Set</code>		
	<code>{foo1}</code>	<code>{foo1, foo2}</code>	<code>{foo1, foo2, foo3}</code>
<code>return this == obj;</code>	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never
<code>Foo f = (Foo) obj; return ((this.x % 2) == (f.x % 2));</code>	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never
<code>Foo f = (Foo) obj; this.x = f.x; return this.x == f.x;</code>	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never	<input type="radio"/> Always <input type="radio"/> Sometimes <input type="radio"/> Never

This page intentionally left (mostly) blank

The exam continues on the next page.

Q4 Field of Hopes and Dreams

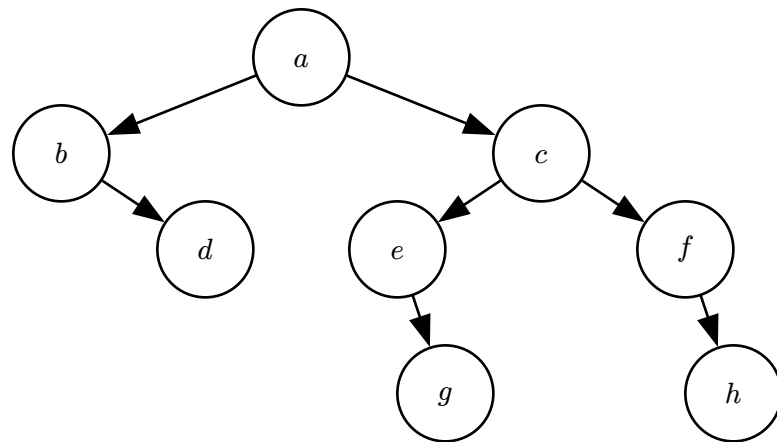
(19 points)

Lancer has built a devious forest maze in the shape of a Binary Search Tree, with an `int` label at each node. As with standard BSTs, there are no duplicate labels.

Q4.1 (6 points)

Below is one of the BST mazes that Lancer could have built, with labels hidden by unknown variables a through h (not in alphabetical order).

For the following logical comparisons, identify whether they are always, sometimes, or never **true** depending on the values of the variables a through h .



$$a > c$$

☐ Always

☐ Sometimes

☐ Never

$$b == d$$

☐ Always

☐ Sometimes

☐ Never

$$d < e$$

☐ Always

☐ Sometimes

☐ Never

$$(b + d) > a$$

☐ Always

☐ Sometimes

☐ Never

$$(c - h) < 0$$

☐ Always

☐ Sometimes

☐ Never

$$(b + d) == 0$$

☐ Always

☐ Sometimes

☐ Never

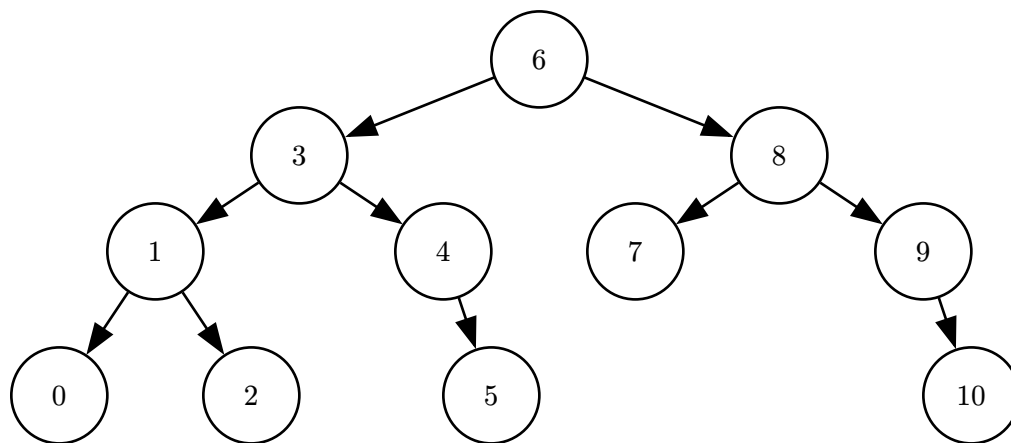
(Question 4 continued...)

Q4.2 (13 points) Lancer builds a **ForestPathNode** class to represent his BST maze.

```
public class ForestPathNode {  
    public int label;  
    public ForestPathNode left;  
    public ForestPathNode right;  
  
    // ... methods not shown ...  
}
```

Suppose that the heroes know the correct **label** of their desired destination **dest**. Build a **ForestPathIterator** that returns the correct path from the input node to the node with the given label, using the directions "**left**" and "**right**".

Below is an example BST forest maze:



- A **ForestPathIterator** that takes in the root node of the above forest maze and a **dest** of 5 should exhaust its elements after **next()** returns "**left**", then "**right**", then "**right**".
- A **ForestPathIterator** that takes in the root node and a **dest** of 8 should exhaust its elements after **next()** returns "**right**".

Assume that the given **dest** is in the maze.

Your solution should construct in $\Theta(1)$ time.

The skeleton code begins on the next page.

(Question 4 continued...)

```
public class ForestPathIterator implements Iterator<_____> {

    private _____ curr;

    private _____ dest;

    public ForestPathIterator(ForestPathNode root, int dest) {
        _____;
        _____;
    }

    @Override
    public boolean hasNext() {
        return _____;
    }

    @Override
    public _____ next() {
        if (_____) {

            curr = _____;

            return _____;
        } else {

            curr = _____;

            return _____;
        }
    }
}
```

Q5 Recycling

(18 points)

Consider the following raw, singly-linked `SLNode` class:

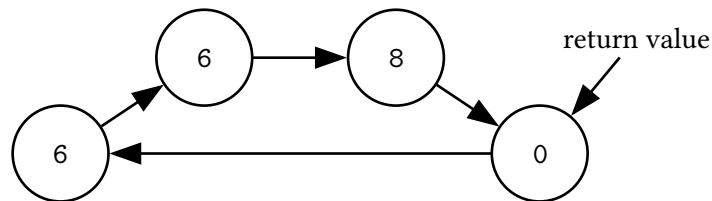
```
public class SLNode<T> {  
    public T item;  
    public SLNode<T> next;  
    public SLNode(T item) {  
        this.item = item;  
        this.next = null;  
    }  
}
```

Q5.1 (8 points)

Complete `cyclify`, which takes in a `int[] arr` and returns a new, *circular*, `SLNode<Integer>` where each node corresponds to an element in the `int[]`. **Return the last node** in the resulting circular linked list.

Assume that `arr` is nonempty.

For example, `cyclify(new int[]{6, 6, 8, 0})` should build the circular linked list below, and return the `SLNode` corresponding to 0.



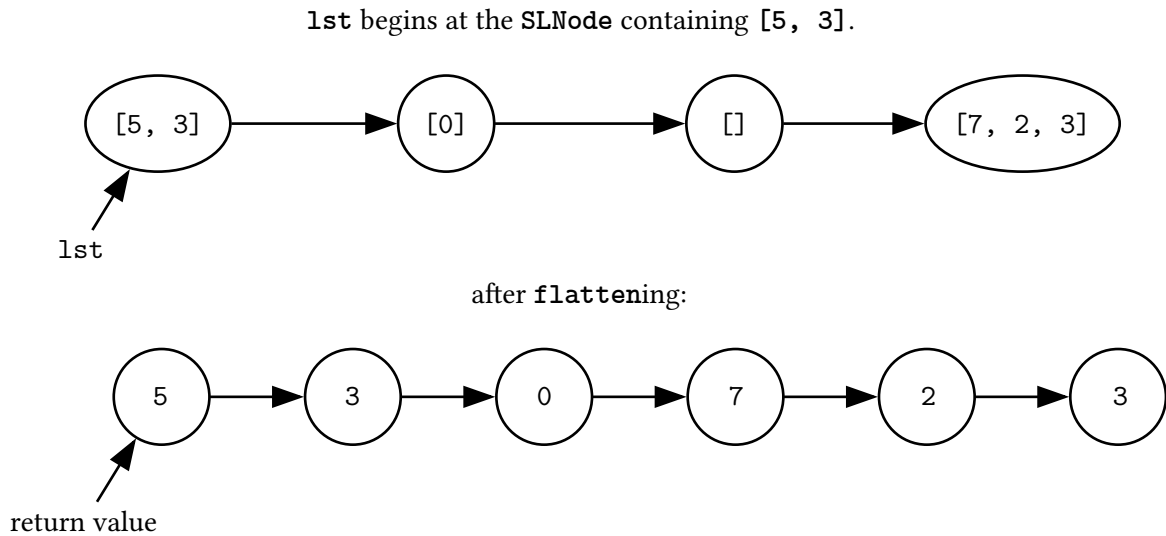
```
public SLNode<Integer> cyclify(int[] arr) {  
    SLNode<_____> curr = _____;  
    SLNode<_____> head = _____;  
    for (int i = _____; _____; _____) {  
        _____;  
        _____;  
    }  
    _____;  
    return curr;  
}
```

(Question 5 continued...)

Q5.2 (10 points)

Complete `flatten`, which takes in a (non-circular) `SLNode<int[]> lst` and flattens it into a (non-circular) `SLNode<Integer>` as shown in the following diagram. You may use `cyclify` as defined in part (a).

An example is given in the following diagram:



```
public SLNode<Integer> flatten(SLNode<int[]> lst) {  
    if (_____) {  
        return null;  
    }  
    SLNode<_____> rest = _____;  
    if (_____) {  
        return rest;  
    }  
    SLNode<_____> lastOfCurr = _____;  
    SLNode<Integer> result = _____;  
    _____;  
    return result;  
}
```

Q6 Critical Thinking

(14 points)

Noah is brainstorming **Ideas** for the midterm, and he wants to know which are bad **Ideas**.

Fortunately, Noah knows there's always a **Critic** who can review any **Idea**:

```
public interface Idea {          public interface Critic {
    // methods not shown        /* Returns how good a given Idea is. */
                                int review(Idea idea);
}                                }
                                }
```

Q6.1 (4 points) Complete **CriticComparator**, which takes in some **Idea**. It compares two **Critics** according to their **review** values of that **Idea** (in natural order).

```
public class CriticComparator implements Comparator<Critic> {
    // Add instance variables (if any needed) here

    _____;

    public CriticComparator(Idea idea) {

        _____;

    }

    @Override
    public int compare(Critic c1, Critic c2) {

        _____;

    }
}
```

(Question 6 continued...)

Q6.2 (10 points)

Complete `IdeaComparator`, which considers an array of `Critics` called `consideredCritics`.

It compares two `Ideas` (in natural order) according to their highest respective `Critic` review across all `Critics` being considered.

You may assume there is at least one `Critic` being considered when `compare` is called.

You may use `CriticComparator` as defined in the previous subpart in addition to any classes or methods on the reference sheet.

```
public class IdeaComparator implements Comparator<Idea> {
    private Critic[] consideredCritics;
    public IdeaComparator() { ... }

    @Override
    public int compare(Idea idea1, Idea idea2) {
        _____;
        _____;
        _____;
        _____;
        return _____;
    }
}
```

Q7 Un-Noah-ble**(20 points)**Q7.1 (4 points) What is the runtime of `naa(N)` in terms of N ?

```

public void naa(int N) {
    int b = 61;
    if (N <= 1) {
        return;
    }
    if (Math.random() <= 0.0001) {
        b = 8;
        naa(N - 1);
    }
    for (int i = 1; i < b; i += 7) {
        System.out.println("yum");
    }
}

```

Best case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Worst case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Q7.2 (4 points) What is the runtime of `melo(N)` in terms of N ?

```

public void melo(int N) {
    if (N <= 1) {
        return;
    }
    melo(N / 2);
    if (N % 2 == 0) {
        melo(N / 2);
    }
}

```

Best case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Worst case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

(Question 7 continued...)

Q7.3 (6 points) What is the runtime of `baco(N)` in terms of N ?

```
public void baco(int N) {
    int[] spaghetti = new int[N];
    spaghetti[0] = Math.round(Math.random());

    for (int i = 0; i < N - 1; i += 1) {
        spaghetti[i + 1] = 0;
        for (int x = 0; x < spaghetti[i]; x += 1) {
            spaghetti[i + 1] += 2;
        }
    }
}
```

Best case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Worst case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Q7.4 (6 points) What is the runtime of `rame(N)` in terms of N ?

```
public void rame(int N) {
    for (int i = 1; i < N; i *= 2) {
        sush(i);
    }
}

private void sush(int i) {
    if (i >= 1) {
        sush(i / 2);
    }
}
```

Best case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Worst case:

- | | | | | |
|--|--|--|--|--|
| <input type="radio"/> $\Theta(1)$ | <input type="radio"/> $\Theta(\log(\log N))$ | <input type="radio"/> $\Theta(\log N)$ | <input type="radio"/> $\Theta((\log N)^2)$ | <input type="radio"/> $\Theta(\sqrt{N})$ |
| <input type="radio"/> $\Theta(N)$ | <input type="radio"/> $\Theta(N \log N)$ | <input type="radio"/> $\Theta(N^2)$ | <input type="radio"/> $\Theta(N^2 \log N)$ | <input type="radio"/> $\Theta(N^3)$ |
| <input type="radio"/> $\Theta(N^3 \log N)$ | <input type="radio"/> $\Theta(2^N)$ | <input type="radio"/> $\Theta(N!)$ | <input type="radio"/> $\Theta(N^N)$ | <input type="radio"/> Infinite loop |

Q8 Am I Worth Nothing to You?

(0 points)

Congrats on finishing the exam! The below questions are just for fun; **nothing on this page is worth any points.**

Q8.1 (0 points) Which people, if any, are telling the truth?

Karen: "If Noah is telling the truth, then I am lying."

Noah: "Karen and Wilson are both liars!"

Wilson: "If Noah is lying, then all of us are lying."

☐ Karen

☐ Noah

☐ Wilson

☐ None of the above

Q8.2 (0 points; one cookie from Dawn) Who is NOT on CS61BL staff this semester?

☐ Samuel

☐ Eric

☐ Erik

☐ Kanav

☐ Kevin

☐ Stacey

☐ Miller

☐ Alonzo

☐ Lawrence

☐ Dennis

☐ Andrew

☐ Benjamin

☐ Gabe

☐ Noelle

☐ Anniyat

☐ David

☐ LeBron

☐ Susie

☐ Stanley

☐ Amanda

☐ Teresa

☐ Curtis

☐ Yinqi

☐ Yashna

☐ Apollo

☐ Rico

☐ Julian

☐ Jonah

☐ Sophia

☐ Circle

Q8.3 (0 points) Leave any feedback, comments, and/or drawings in the box below!