You have 60 minutes.

Show your work and explain your answers. Good work often earns partial credit. A correct answer with no explanation often earns little or no credit.

If you are asked to write code but you do not know the exact Python required, then try to write code that is approximately correct. If you think that your code does not demonstrate that you understand the solution, then describe your idea in English as well. Be precise enough that I cannot misinterpret your solution.

Good luck.

I'm going to ask you two unrelated questions about our `gcd` function from class:

```
def gcd(a, b):
    """The arguments a and b are integers with a >= b >= 0. Returns the greatest
    common divisor of a and b."""
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

First, what happens if we accidentally call `gcd` with `b` greater than `a` (and both still nonnegative)?

Second, let m be the number of decimal digits in `a`, and let n be the number of decimal digits in `b`. Using big-$\mathcal{O}$ notation, describe the running time of `gcd` in terms of m and/or n.

THIS PART HAS BEEN CANCELLED; DO NOT DO IT.

Perform a "queue-based radix sort" on the following list of numbers, showing each intermediate result along the way. `[313, 7, 21, 3241, 3420, 27]`

Perform a "stack-based radix sort" on the same list of numbers, showing each intermediate result along the way. `[313, 7, 21, 3241, 3420, 27]`

Remember the `BetterSet` class, which was a subclass of `Set`? I just remembered another method that I wanted for `BetterSet`. Please implement this method recursively.

```
def subset(self, f):
    """The argument f is a predicate (a function of one argument, that outputs
    either True or False). Returns a new BetterSet, consisting of all of the
    elements in this set for which f is True. This set is left unaltered."""
```

Using big-$\mathcal{O}$ notation, describe the running time of `subset`.

Consider the following implementation of `List`. As usual, there is a big linked list, based at `self.head`, containing all of the data nodes, and there is a `self.count` member that tracks the length n of the data list. But there is a second linked list, based at `self.shortcuts`, of length $\sqrt{n}$. (For simplicity, assume that n is a perfect square, so that $\sqrt{n}$ is an integer.) Each node in this shortcut list contains, as its data element, a pointer to a node in the big linked list. These shortcut pointers are spaced evenly throughout the big linked list. For example, if $n = 100$, then the shortcut list has 10 nodes, which point to nodes $0, 10, 20, ..., 80, 90$ of the data list.

Implement `__getitem__`. Do not include a docstring.

Using big-$\mathcal{O}$ notation, describe the running time of `__getitem__`.

In this question, an $m \times n$ *grid* is a `List` of m items, each of which is a `List` of n numbers. For example, `((3, -1, 2.1), (14, 3.25, 3))` is a 2×3 grid. You are writing a program that contains a grid `g` of indeterminate size. Using `listMap`, `listFilter`, and `listFold`, find the smallest number in `g`. (You may not need to use all three functions. However, you should try to get as much work done using these functions as you can, rather than writing your own custom code. My solution fits in one line.)

Using big-$\mathcal{O}$ notation, describe how much time that computation takes.