You have 70 minutes.

Show your work and explain all of 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.

When you are asked to implement an operation or data structure, it is understood that efficient implementations earn more credit than inefficient ones.

When you are asked to describe how efficient an algorithm or data structure is with regard to time or space, you should include a big-O estimate, as well as any other useful descriptive information.

Good luck.

1. I want a data structure `RoundRobin` that describes a system in which objects (people, computers, programs, or whatever) take turns. For example, if Jan, Juan, and Joan are in the system, then the turns go Jan, Juan, Joan, Jan, Juan, Joan, etc. At any time I can add a new object to the system; when I do, everybody already in the system takes her turn as usual, before the new object gets its first turn. The most important methods of `RoundRobin` are `next()`, which returns the object whose turn it is (or `None` if the system is empty), and `add()`, which adds a new object to the mix.

In words, pictures, and/or Python code, describe precisely how you would implement `next()` and `add()`. How fast is each operation?

2. Every application written for Mac OS X contains an `Info.plist` XML file that specifies metadata such as which kinds of files the application is able to edit. The following XML tags were taken from TextWrangler's `Info.plist`. Suppose that I use a stack to check the well-formedness of the sequence of XML tags given below. Draw the stack *when it is at its deepest*.

```
plist, dict, key, /key,
string, /string, key, /key,
array, dict, key, /key,
array, string, /string, string,
/string, /array, key, /key,
string, /string, /dict, /array,
key, /key, array, dict,
key, /key, string, /string,
key, /key, array, string,
/string, /array, /dict, /array,
key, /key, true/, /dict,
/plist
```

3. In our Efficiency assignment we implemented `Array` using a linked list. Some people endowed `Array` with `head` and `tail` pointers, so that both appending and prepending were fast. However, accessing elements in the middle of the array was still slow. Here's an idea: In addition to `head` and `tail`, let's add a `middle` pointer, that points at a node near the middle of the array. Then we can get fast access to the middle, too. Right?

Assuming we could implement this idea, how much would it improve the speed of accessing elements throughout the array?

4. Here's a more ambitious idea. Instead of three pointers `head`, `tail`, and `middle`, I have about $\sqrt{n}$ pointers (for an array of n elements), pointing at nodes spaced evenly throughout the array, including the first and last nodes. These $\sqrt{n}$ pointers are themselves stored in a linked list; my `Array` data structure stores only a pointer to the first node of this linked list of $\sqrt{n}$ pointers into the big linked list of n data nodes. Assuming we could implement this, how much would it improve the speed of accessing elements throughout the array?

5. Suppose I have a function `num()` that takes in any digit and returns the corresponding integer. For example, `num('3')` returns the integer 3. For another example, `num('32')` doesn't work at all, because `'32'` is not a digit but rather a multi-digit numeral.

Write a Python function `number()` that takes in a multi-digit numeral, uses `num()` and recursion, and returns the integer corresponding to the given numeral. (You do not need to handle decimal points or negation signs.) Also draw the call stack, when it is at its deepest, for the call `number('32767')`.

6. I want to change the `Set` ADT by adding a new method, `unite()`. When you make the call `a.unite(b)`, the set `a` absorbs all of the elements of the set `b`. This operation is surprisingly useful in applications of set data structures. In these applications, the set `b` is never again used on its own, once it has been absorbed into another set `a`; you may assume this.

In words, pictures, and/or Python code, describe precisely how you would implement `Set`, including `unite()`. How fast is `unite()`?