Data Structures

Tying Up Loose Ends

D.S. Blank

Fall 2010

Data Structures

- Object Oriented Programming
- Sorting

Object Oriented Programming

- Suppose that you want an object almost like another, but want to “extend” it
- Consider that you have a Student class, but now you want to have a Teacher class with additional methods
- What to do?

Object Oriented Programming

- Suppose that you want an object almost like another, but want to “extend” it
- Consider that you have a Student class, but now you want to have a Teacher class with additional methods
- What to do?
  - Use Inheritance (sometimes called subclass, or derived class)

OOP Inheritance

- Ability to reuse code, without duplication
- Can create specialized variants (Teacher, Student, Advisor, TeachersAssistant, etc)
- Start with a “base class”
- You can “override” base methods
- You can “extend” with new methods
- You can haver layers and layers of derived classes

OOP Inheritance: Override

```
class Person:
    def get_access(self):
        return [ ]

class Student(Person):
    def get_access(self):
        return [“Read”]

class Teacher(Student):
    def get_access(self):
        return [“Read”, “Write”]
```

```
>>> p = Person()
>>> s = Student()
>>> t = Teacher()
>>> p.get_access()
[ ]
>>> s.get_access()
[“Read”]
>>> p.get_access()
[“Read”, “Write”]
```

OOP Inheritance: Extend

```
class Person:
    def get_access(self):
        return [ ]

class Teacher(Person):
    def __init__(self):
        self.access = [ ]

    def get_access(self):
        return self.access

    def add_access(self, v):
        self.access.append(v)
```

```
>>> p = Person()
>>> t = Teacher()
>>> p.get_access()
[ ]
>>> t.get_access()
[ ]
>>> t.set_access(“Eat”)
>>> t.get_access()
[“Eat”]
```

OOP: Multiple Inheritance

- Inheritance from two or more classes
- Not all languages have Multiple Inheritance
  - Some have Interfaces instead
    - Java, C# have multiple interfaces
- Python allows Multiple Inheritance
  - Need to know how Python resolves a method's definition
  - Can be tricky, and prone to errors

OOP Inheritance: MI

```
class Person:
    def get_access(self):
        return [ ]

class Operator:
    def get_access(self):
        return [“Call”]

class Student(Person, Operator):
    pass

s = Student()
s.get_access()
```

Sorting: Comparing Run Time

- There are many different algorithms for sorting, and each can use different amounts of space and time
- Consider the following:

```
def sort(list):
    for i in range(len(list) – 1):
        for j in range(i, len(list)):
            if list[i] > list[j]:
                list[i], list[j] = list[j], list[i]
```

We would like a way to say how much time it takes.

Big O Notation

- Captures the upper bound of worst case
- Is a dominating function
- Multipliers aren't important
- Examples:
  - O(1) – Constant time
  - O(n) – Linear time
  - O(n ** 2) - quadratic
  - O(2 ** n) - exponential

What is the Big O?

```
def bubble_sort(list):
    for j in range(len(list) - 1, 1, -1):
        swap = False
        for i in range(j):
            if list[i] > list[i + 1]:
                list[i], list[i + 1] = list[i + 1], list[i]
                swap = True
        if not swap:
            break
```