[320] Object Oriented Programming

Tyler Caraza-Harter

Review Complexity

Unless otherwise specified, what kind of complexity analysis is expected?

1. worst case
2. best case
3. average case

When analyzing algorithm complexity, what does **f(N)** usually represent?

To show **f(N)** $\in$ **O(N^3)**, we need to show that the **y=f(N)** curve is under some **y=????*N^3** curve. What advantages do we have to make this easier?

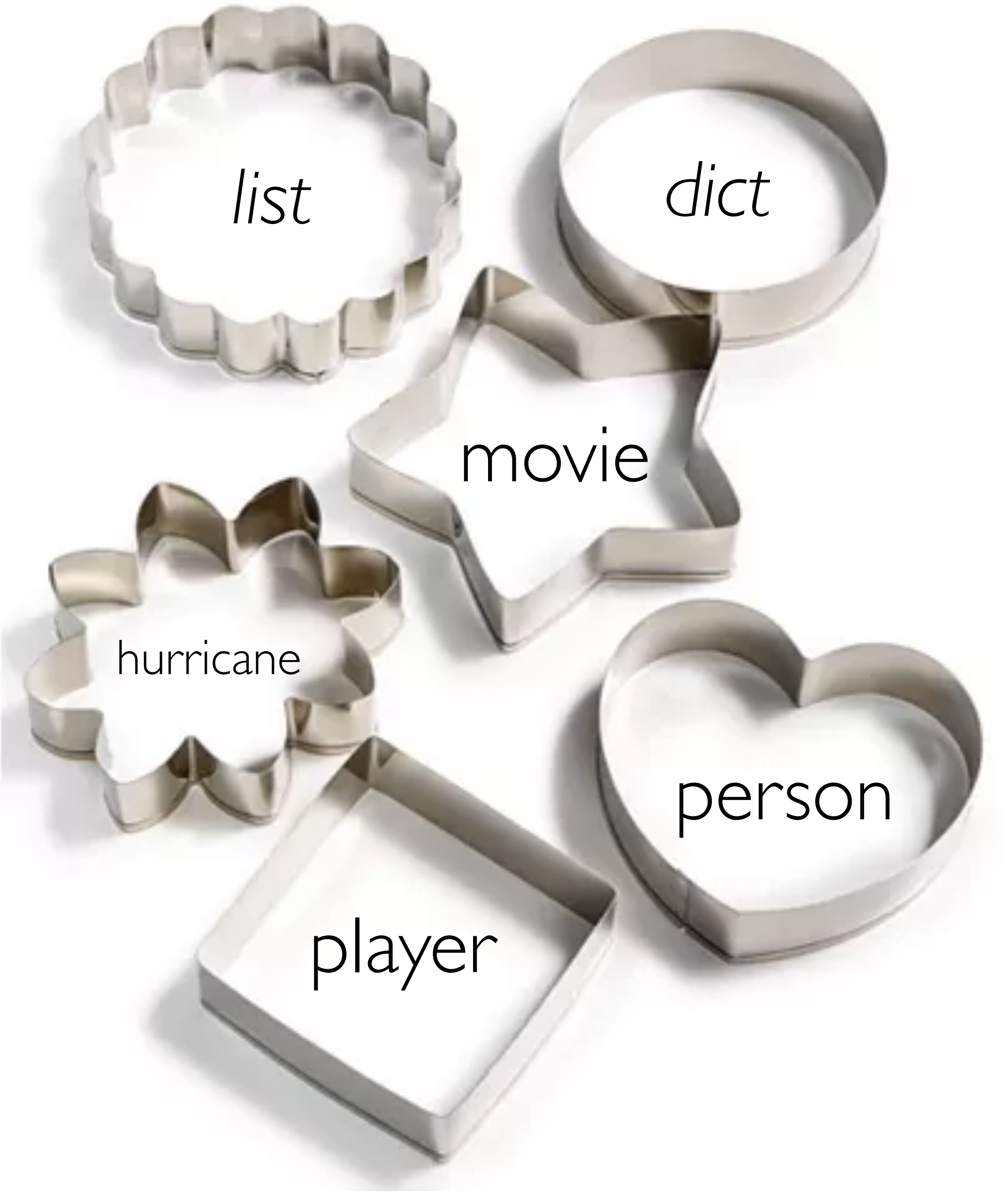
1. replace ???? with N
2. replace ???? with a constant
3. ignore small N values

True or False: **F(N) = N+(N-1)+(N-2)+2+1** is in **O(N)** because we can can throw away the non-leading terms.

O(????**)** is better than **O(N)**, but worse than **O(1)**

Creating New Types

Classes and Other Types



https://www.macys.com/shop/product/martha-stewart-collection-set-of-6-cookie-cutters-created-for-macys?ID=5467270

Objects

```python
from collections import namedtuple
```

need to import this data struct

name of that type

creates a new type!

name of that type

```python
Person = namedtuple("Person", ["fname", "lname", "age"])
```

New types in CS 220/301

```python
from collections import namedtuple
```

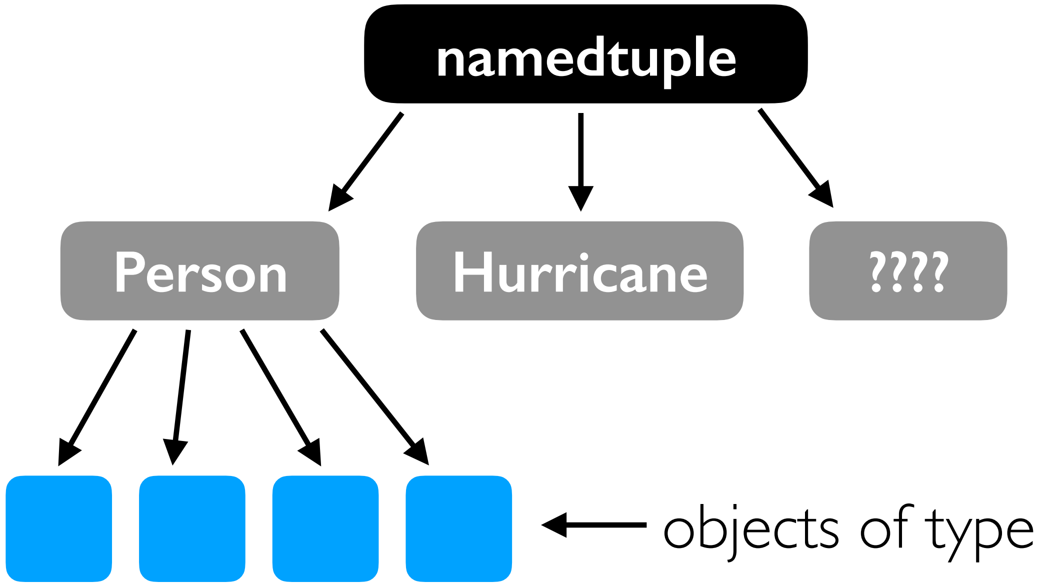
need to import this data struct

name of that type

creates a new type!

name of that type

```python
Person = namedtuple("Person", ["fname", "lname", "age"])
```



objects of type Person

```
from collections import namedtuple
```

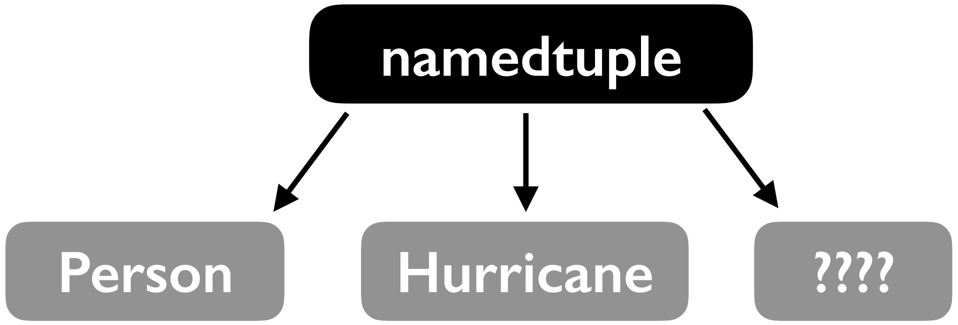
need to import this data struct

name of that type

creates a new type!

name of that type

```
Person = namedtuple("Person", ["fname", "lname", "age"])
```



namedtuple

Person

Hurricane

????

```
p = Person("Alice", "Anderson", 30)
```

creates a object of type Person (sub type of namedtuple)
(like **str(3)** creates a new string or **list()** creates a new list)

```
print("Hello " + p.fname + " " + p.lname)
```

```python
from collections import namedtuple

Person = namedtuple("Person", ["fname", "lname", "age"])
```

what is the difference?

```python
from recordclass import recordclass

Person = recordclass("Person", ["fname", "lname", "age"])
Person = namedtuple("Person", ["fname", "lname", "age"])
p = Person("Alice", "Anderson", 30)
```

```
print("Hello " + p.fname + " " + p.lname)
```

✓

namedtuple
types have
attributes

```
print("Hello " + p.get_full_name())
```

✗

the don't have
methods

classes are a way to create new types of
objects with both **attributes** and **methods**

Attributes

```
class Person:
    pass
```

create a Person type/class

```
p1 = Person()
p2 = Person()
p3 = Person()
```

create some objects of type Person

set some attributes

```
p1.Fname = "Joseph"
p2.fname = "Sacha"
p3.fname = "Shri Shruthi"
```

Objects created from classes are mutable.
Attribute names are not fixed at creation.

Attribute Names/Values are like Keys/Values

Using Dict	Using `class Point:` `pass`	
`d = dict()`	`p = Point()`	`p = Point()`

`d["x"] = 3`
`d["y"] = 4`
	`setattr(p, "x", 3)`
	`setattr(p, "y", 4)`
		`p.x = 3`
		`p.y = 4`

`tot = d["x"] +d["y"]`	`tot = (getattr(p, "x")`
	`+getattr(p, "y"))`
		`tot = p.x + p.y`

`has_z = "z" in d`	`has_z = hasattr(p, "z")`	`# no equivalent`
	avoid this	preferred

only use attribute
names that could also
be variables names

Coding Examples: Animal Classes

Principals

- methods
- checking object type
- type-based dispatch
- self
- constructors