Classes of algorithms

- Iterative algorithms allow us to do more complex things than simple arithmetic
- We can repeat a sequence of steps multiple times based on some decision; leads to new classes of algorithms
- One useful example are “guess and check” methods

Guess and check

- Remember our "declarative" definition of square root of `x`
- If we could guess possible values for square root (call it `g`), then can use definition to check if `g*g = x`
- We just need a good way to generate guesses

Finding a cube root of an integer

- One way to use this idea of generating guesses in order to find a cube root of `x` is to first try `0**3`, then `1**3`, then `2**3`, and so on
- Can stop when reach `k` such that `k**3 > x`
- Only a finite number of cases to try

Some code

```
x = int(raw_input('Enter an integer: '))
ans = 0
while ans**3 < x:
    ans = ans + 1
if ans**3 != x:
    print(str(x) + ' is not a perfect cube')
else:
    print('Cube root of ' + str(x) + ' is '
  + str(ans))
```

<- Find value where ans^3 >= x

<- Check

Extending scope

- Only works for positive integers
- Easy to fix by keeping track of sign, looking for solution to positive case

Some code

```
x = int(raw_input('Enter an integer: '))
ans = 0
while ans**3 < abs(x):
    ans = ans + 1
if ans**3 != abs(x):
    print(str(x) + ' is not a perfect cube')
else:
    if x < 0:
        ans = - ans
    print('Cube root of ' + str(x) + ' is '
  + str(ans))
```

Loop characteristics

- Need a loop variable
  - Initialized outside loop
  - Changes within loop
  - Test for termination depends on variable
- Useful to think about a **decrementing function**
  - Maps set of program variables into an integer
  - When loop is entered, value is non-negative
  - When value is <= 0, loop terminates, and
  - Value is decreased every time through loop
- Here we use `abs(x) – ans**3`

What happens if we miss a condition?

- Suppose we don't initialize the variable?

Remove ans = 0

- Suppose we don't change the variable inside the loop?

Remove ans = ans + 1

Exhaustive enumeration

- Guess and check methods can work on problems with a finite number of possibilities
- Exhaustive enumeration is a good way to generate guesses in an organized manner