# Testing

- Goal:
  - Show that bugs exist
  - Would be great to prove code is bug free, but generally hard
    - Usually can't run on all possible inputs to check
    - Formal methods sometimes help, but usually only on simpler code

Billions of tests

# Test suite

- Want to find a collection of inputs that has high likelihood of revealing bugs, yet is efficient
  - Partition space of inputs into subsets that provide equivalent information about correctness
    - Partition divides a set into group of subsets such that each element of set is in exactly one subset
  - Construct test suite that contains one input from each element of partition
  - Run test suite

# Example of partition

```
def isBigger(x, y):
    """Assumes x and y are ints
       returns True if x is less than y
       else False"""
```

- Input space is all pairs of integers
- Possible partition
  - x positive, y positive
  - x negative, y negative
  - x positive, y negative
  - x negative, y positive
  - x = 0, y = 0
  - x = 0, y != 0
  - x != 0, y = 0

# Why this partition?

- Lots of other choices
  - E.g., x prime, y not; y prime, x not; both prime; both not

irrelevant to problem

- Space of inputs often have natural boundaries
  - Integers are positive, negative or zero
  - From this perspective, have 9 subsets
    - Split x = 0, y != 0 into x = 0, y positive and x =0, y negative
    - Same for x != 0, y = 0

# Partitioning

- What if no natural partition to input space?
  - Random testing – probability that code is correct increases with number of trials; but should be able to use code to do better
  - Use heuristics based on exploring paths through the specifications – **black-box testing**
  - Use heuristics based on exploring paths through the code – **glass-box testing**