Black-box testing

CS314

SUDIPTO GHOSH, GERI GEORG

Input domains

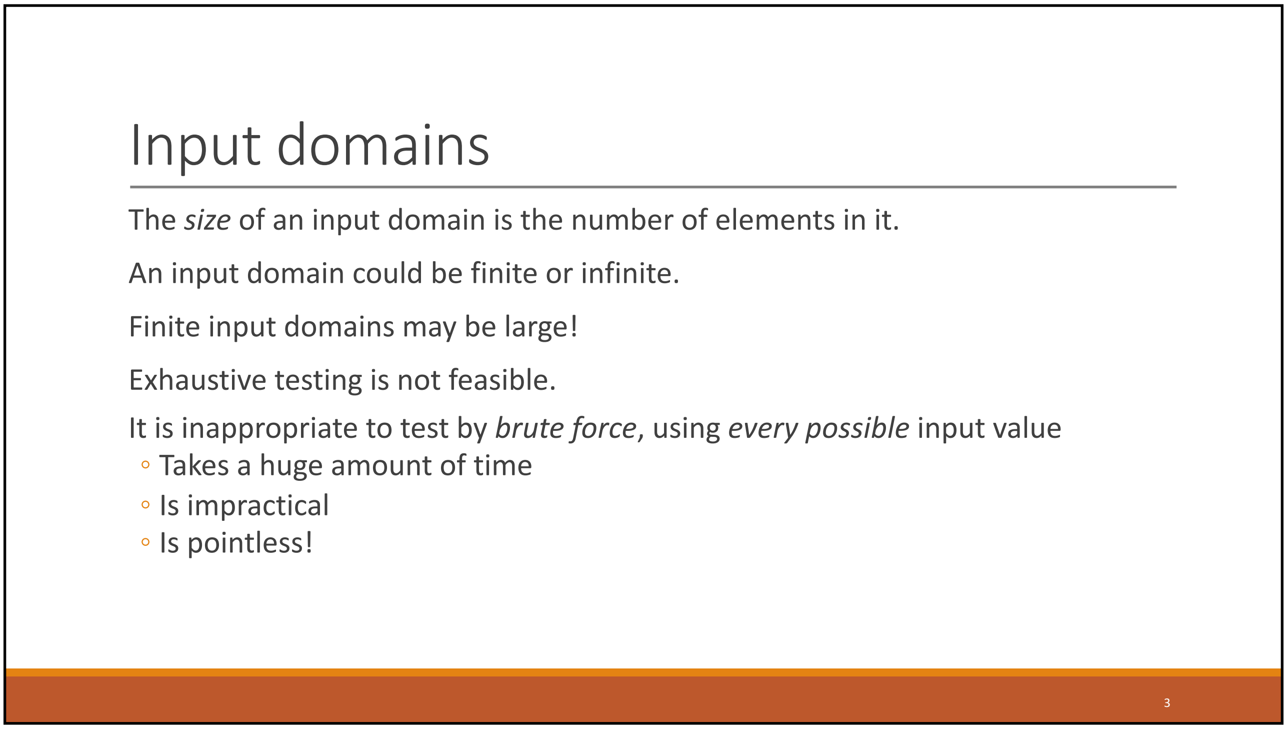
Identify the input domain of *P*.

- The set of all *valid* inputs that a program *P* can expect is the input domain of *P*.

Execute *P* against each element of the input domain.

For each execution of *P*, check if *P* generates the correct output as per its specification *S*.

<<Class exercise – example from sorting>>



Input domains

The *size* of an input domain is the number of elements in it.

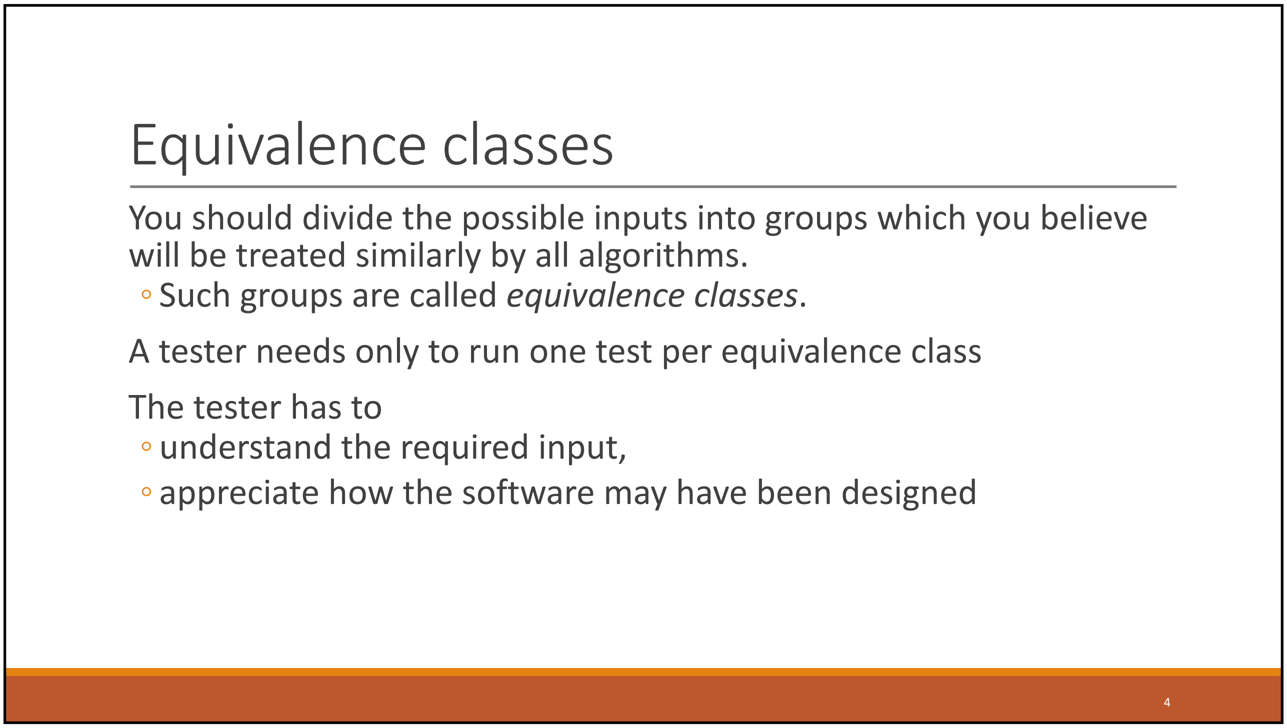
An input domain could be finite or infinite.

Finite input domains may be large!

Exhaustive testing is not feasible.

It is inappropriate to test by *brute force*, using *every possible* input value

- Takes a huge amount of time
- Is impractical
- Is pointless!



Equivalence classes

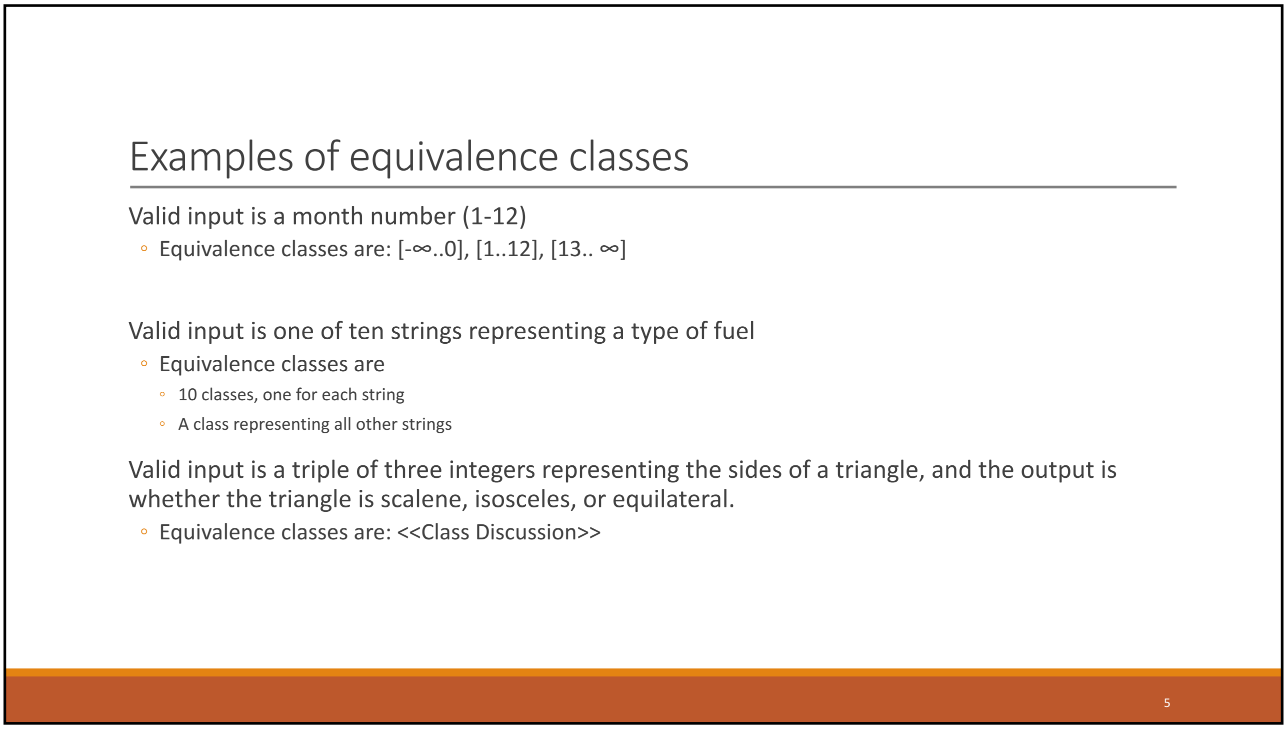
You should divide the possible inputs into groups which you believe will be treated similarly by all algorithms.

- Such groups are called *equivalence classes*.

A tester needs only to run one test per equivalence class

The tester has to

- understand the required input,
- appreciate how the software may have been designed



Examples of equivalence classes

Valid input is a month number (1-12)

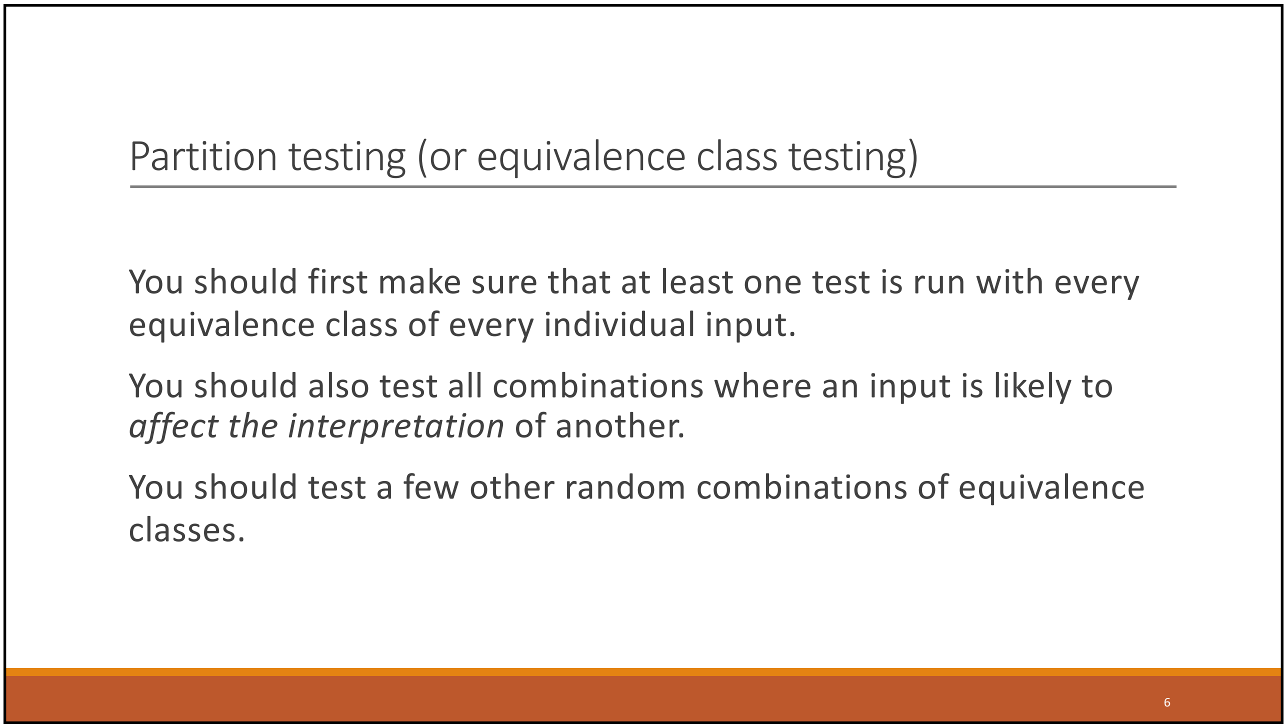
- Equivalence classes are: [-∞..0], [1..12], [13.. ∞]

Valid input is one of ten strings representing a type of fuel

- Equivalence classes are
  - 10 classes, one for each string
  - A class representing all other strings

Valid input is a triple of three integers representing the sides of a triangle, and the output is whether the triangle is scalene, isosceles, or equilateral.

- Equivalence classes are: <<Class Discussion>>

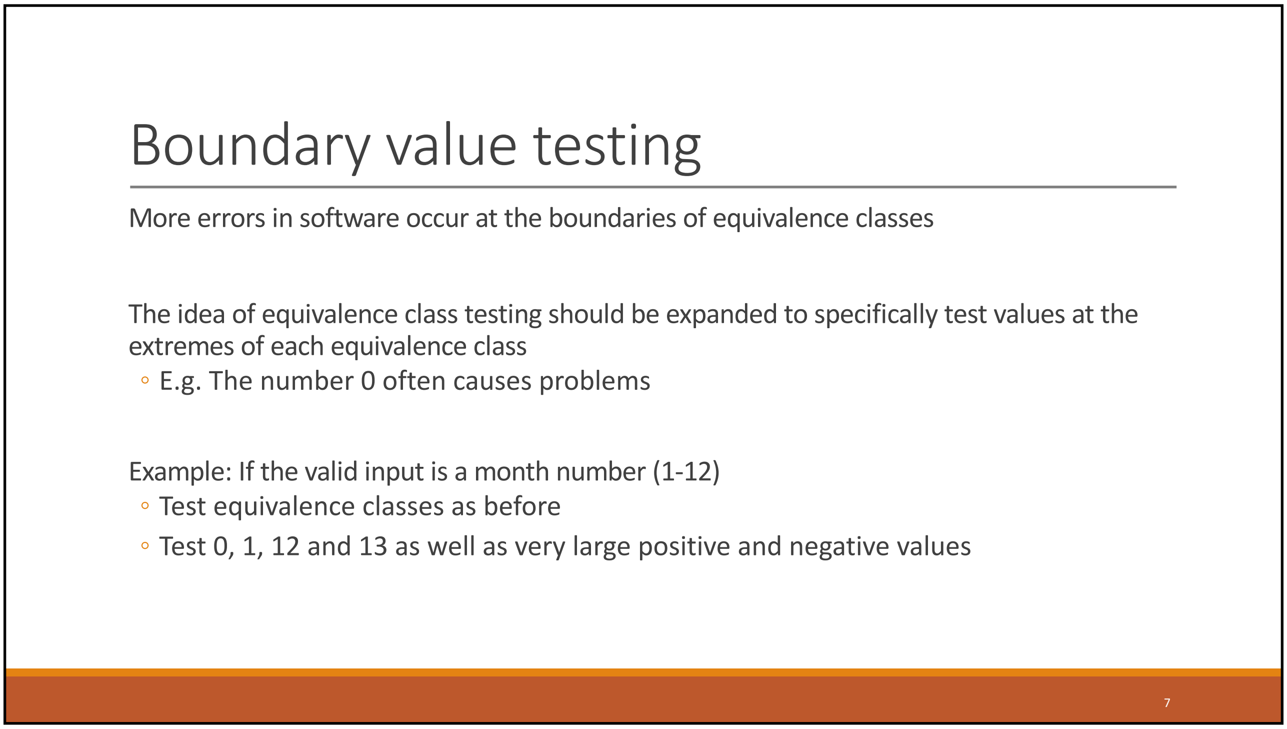


Partition testing (or equivalence class testing)

You should first make sure that at least one test is run with every equivalence class of every individual input.

You should also test all combinations where an input is likely to *affect the interpretation* of another.

You should test a few other random combinations of equivalence classes.



Boundary value testing

More errors in software occur at the boundaries of equivalence classes

The idea of equivalence class testing should be expanded to specifically test values at the extremes of each equivalence class

- E.g. The number 0 often causes problems

Example: If the valid input is a month number (1-12)

- Test equivalence classes as before
- Test 0, 1, 12 and 13 as well as very large positive and negative values