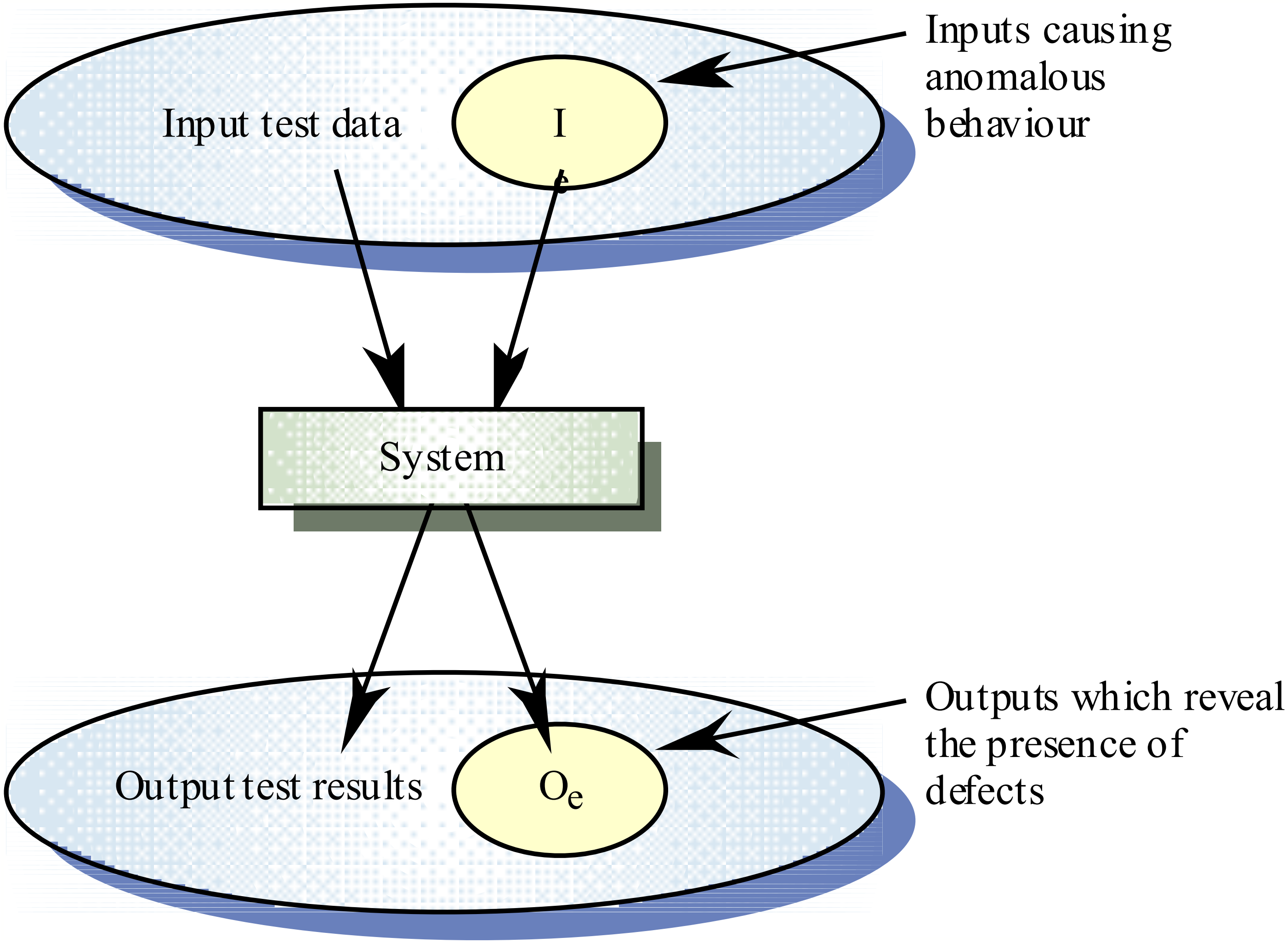
Software Testing

Part 3 of 4

Black-box Testing

- An approach to testing where the program is considered as a ‘black-box’
- The program test cases are based on the system specification
- Test planning can begin early in the software process

Black-box testing



Input test data

I
e

Inputs causing anomalous behaviour

System

Output test results

O_e

Outputs which reveal the presence of defects

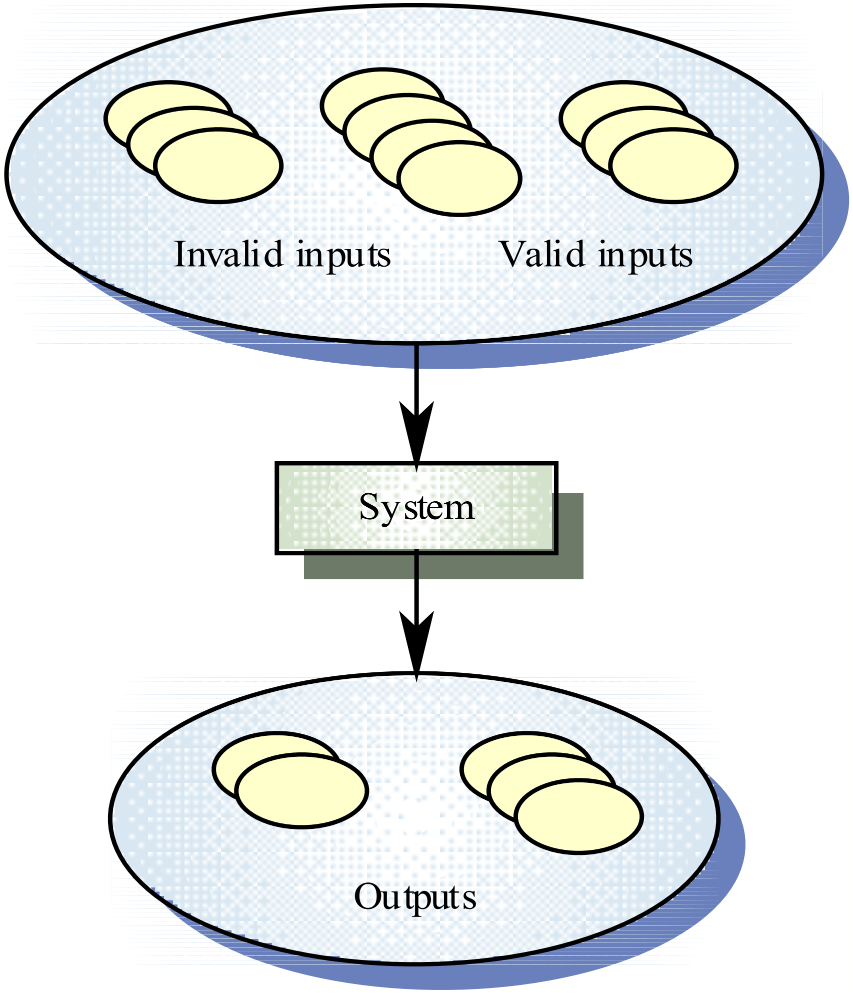
Pairing Down Test Cases

- Use methods that take advantage of symmetries, data equivalencies, and independencies to reduce the number of necessary test cases.
  - Equivalence Testing
  - Boundary Value Analysis
- Determine the ranges of working system
- Develop equivalence classes of test cases
- Examine the boundaries of these classes carefully

Equivalence Partitioning

- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition

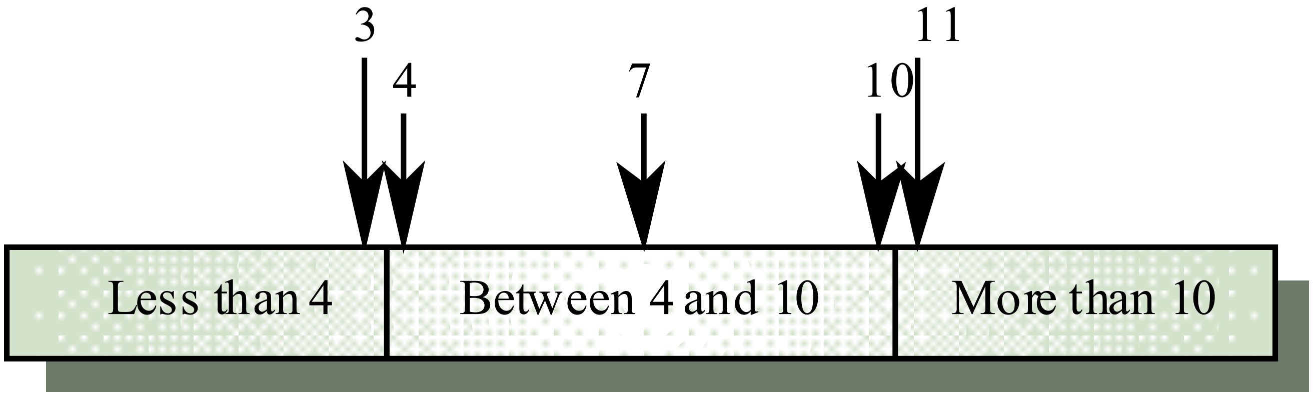
Equivalence Partitioning



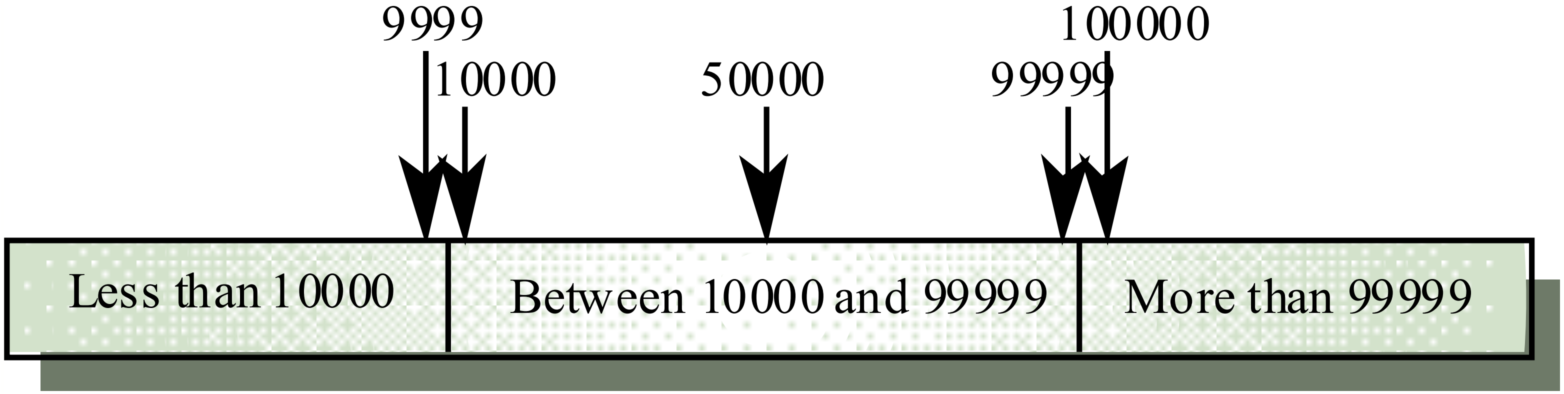
Boundary Value Testing

- Partition system inputs and outputs into “equivalence sets”
  - If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are < 10,000, 10,000 - 99, 999 and > 10, 000
- Choose test cases at the boundary of these sets
  - 00000, 09999, 10000, 99999, 10001

Equivalence Partitions



Number of input values



Input values

Search Routine Specification

```
procedure Search (Key : ELEM ; T: ELEM_ARRAY;
     Found : in out BOOLEAN; L: in out ELEM_INDEX) ;

Pre-condition
          -- the array has at least one element
          T’FIRST <= T’LAST
Post-condition
          -- the element is found and is referenced by L
          ( Found and T (L) = Key)
or
          -- the element is not in the array
          ( not Found and
          not (exists i, T’FIRST >= i <= T’LAST, T (i) = Key ))
```

Search Routine - Input Partitions

- Inputs which conform to the pre-conditions
- Inputs where a pre-condition does not hold

- Inputs where the key element is a member of the array

- Inputs where the key element is not a member of the array

Testing Guidelines - Sequences

- Test software with sequences which have only a single value
- Use sequences of different sizes in different tests
- Derive tests so that the first, middle and last elements of the sequence are accessed
- Test with sequences of zero length

Search Routine - Input Partitions

Array	**Element**
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Sorting Example

- Example: sort (lst, n)
  - Sort a list of numbers
  - The list is between 2 and 1000 elements

- Domains:
  - The list has some item type (of little concern)
  - n is an integer value (sub-range)
- Equivalence classes;
  - n < 2
  - n > 1000
  - 2 <= n <= 1000

Sorting Example

- What do you test?
- Not all cases of integers
- Not all cases of positive integers
- Not all cases between 1 and 1001

- Highest payoff for detecting faults is to test around the boundaries of equivalence classes.

- Test n=1, n=2, n=1000, n=1001, and say n= 10
- Five tests versus 1000.

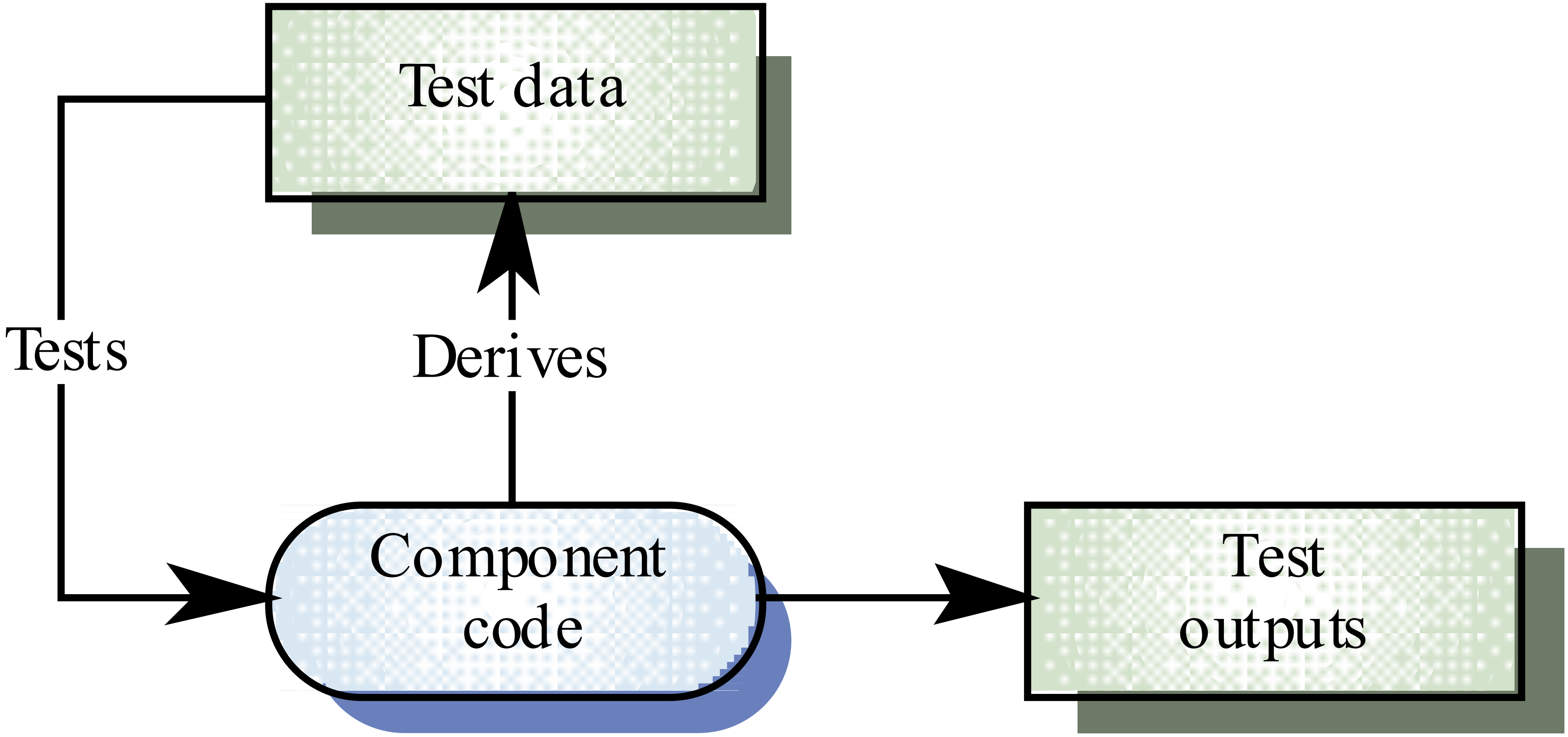
White-box Testing

- Sometime called structural testing or glass-box testing
- Derivation of test cases according to program structure
- Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)

Types of Structural Testing

- Statement coverage -
  - Test cases which will execute every statement at least once.
  - Tools exist for help
  - No guarantee that all branches are properly tested. Loop exit?
- Branch coverage
  - All branches are tested once
- Path coverage - Restriction of type of paths:
  - Linear code sequences
  - Definition/Use checking (all definition/use paths)
  - Can locate dead code

White-box testing



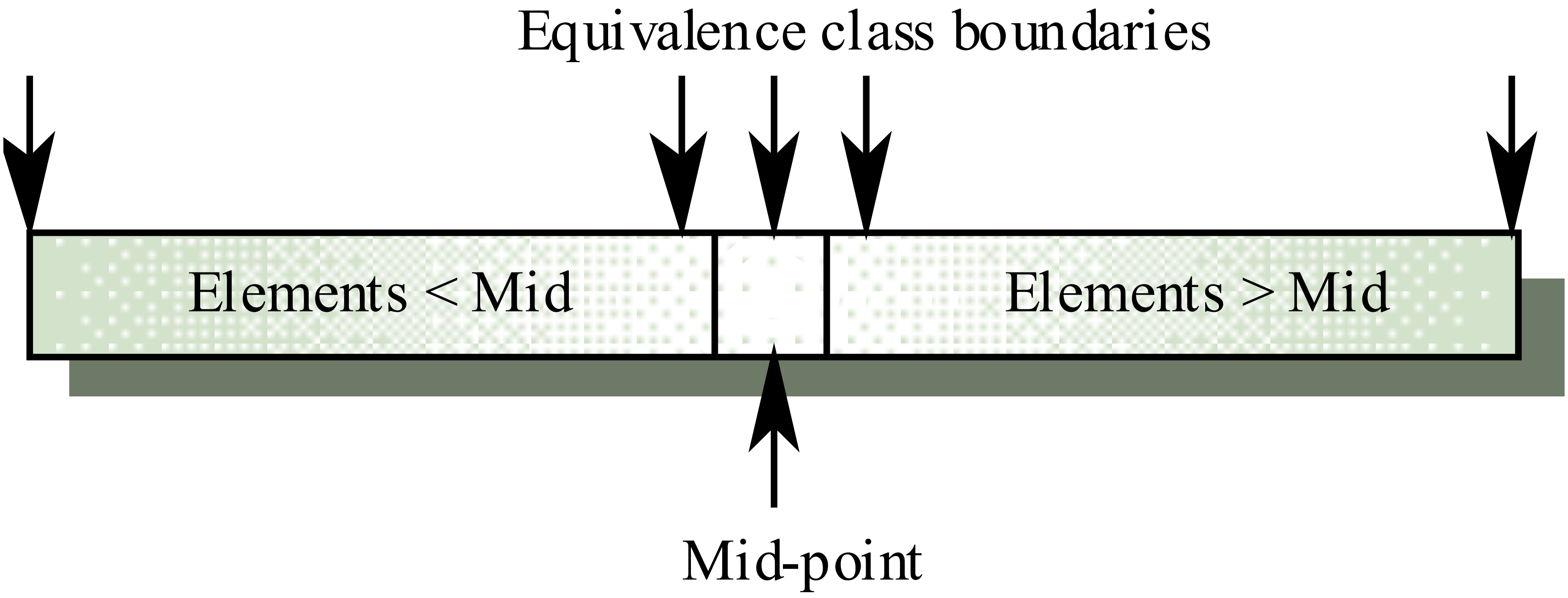
White Box Testing - Binary Search

```
int search ( int key, int [] elemArray)
{
   int bottom = 0;
   int top = elemArray.length - 1;
   int mid;
   int result = -1;
   while ( bottom <= top )
   {
       mid = (top + bottom) / 2;
       if (elemArray [mid] == key)
       {
         result = mid;
         return result;
       } // if part
       else
       {
         if (elemArray [mid] < key)
            bottom = mid + 1;
         else
            top = mid - 1;
       }
   } //while loop
   return result;
} // search
```

Binary Search Equivalence Partitions

- Pre-conditions satisfied, key element in array
- Pre-conditions satisfied, key element not in array
- Pre-conditions unsatisfied, key element in array
- Pre-conditions unsatisfied, key element not in array
- Input array has a single value
- Input array has an even number of values
- Input array has an odd number of values

Binary Search Equivalence Partitions



Binary Search - Test Cases

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??