Finding an item in a sequence

Search for item \( n \) in a sequence, \( seq \).
Return True if \( n \) is a member, else False.

(\text{Python \texttt{in} operator})
Finding an item in a sequence

def isIn(seq, t):
    """(sequence, item) -> boolean
    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
    >>> isIn('hello', 'i')
    False
    >>> isIn([10, 20, 30, 40, 50, 60, 70, 80, 90], 80)
    True
    ""
    found = False
    for item in seq:
        if item == t:
            return True
    return False

Binary Search: efficient search technique,
as long as the list is already sorted.

Each step divides the remaining data into equal parts and discards one part:
If remaining part is empty, then done (not found).
Go to mid-point of remaining part and compare to target.  
If mid-point is the target, then done (found).
Otherwise, keep the part of the list where n could be, and search that.  (Discard the rest.)
For example, 

```
nums = (1, 3, 4, 6, 8, 9, 11)
target value: 4
```

compare x to 6; x is smaller, so repeat with (1, 3, 4)

```
nums = (1, 3, 4, 6, 8, 9, 11, 13, 15, 25, 99, 100, 102)
target value: 42
```

compare x to 11; x is larger, so repeat with (13, 15, 25, 99, 100, 102)

```
nums = (1, 3, 4, 6, 8, 9, 11, 13, 15, 25, 99, 100, 102)
target value: 42
```

compare x to 99; x is smaller, so repeat with (13, 15, 25)

```
nums = (1, 3, 4, 6, 8, 9, 11, 13, 15, 25, 99, 100, 102)
target value: 42
```

compare x to 15; x is larger, so repeat with (25)

compare x to 25; not equal; done; False is returned

```
Sequential search O(n)
Binary search O(log n)
```

When n is 150 ...

```
150 v. 8
```

When n is around 1000 ...

```
1000 v. 10
```

When n is around 1,000,000 ...

```
1,000,000 v. 20
1,000,000,000 v. 30
```

```
"Big O" notation for general, worst-case scenario
```

```
Check even or odd O(1)
Binary search O(log n)
Sequential search O(n)
"Long" multiplication O(n²)
Match passwords O(2^n)
```