```
LinearSearch(int[] A, int n, int key)
    pos = -1
    for i = 0 to (n-1)
          if (A[i] == key)
          {
                pos = i
                break
           }
    if pos == -1
          print "Not found!"
    else
          print "Found!"
    return pos
```

A = { 1, 7, 4, 3, 9, 6, 2 }

key = 5 ---> Not found - failed!

key = 4 ---> Found, at index 2!

Worst case time complexity: O(n)

Best case time complexity: O(1)

A = { 1, 3, 4, 6, 8, 9, 11 }
key = 4

Comparison 1: key < 6

A' = { 1, 3, 4 }
key = 4

Comparison 2: key > 3

A'' = { 4 }
key = 4

Comparison 3: key == 4

A = { 1, 3, 4, 6, 8, 9, 11 }
key = 8

Comparison 1: key > 6

A' = { 8, 9, 11 }
key = 8

Comparison 2: key < 9

A'' = { 8 }
key = 8

Comparison 3: key == 4

Algorithm BinarySearch$(A, k, \text{low}, \text{high})$:

Input: An ordered array, A, storing n items, whose keys are accessed with method key(i) and whose elements are accessed with method elem(i); a search key k; and integers low and high

Output: An element of A with key k and index between low and high, if such an element exists, and otherwise the special element ***null***

```
if low > high then
    return null
else
    mid ← ⌊(low + high)/2⌋
    if k = key(mid) then
        return elem(mid)
    else if k < key(mid) then
        return BinarySearch(A, k, low, mid − 1)
    else
        return BinarySearch(A, k, mid + 1, high)
```

Worst case time complexity: O(log n)

A = { 1, 3, 4, 6, 8, 9, 11 }
key = 2

Comparison 1: key < 6

A' = { 1, 3, 4 }
key = 2

Comparison 2: key < 3

A'' = { 1 }
key = 2

Comparison 3: key > 1

A''' = {}
Return "Not found"

```
BinarySearch( A, key, low, high )

if (low > high)          // if A is an empty array
      return (-1)        // "Not found"

mid <-  ⌊ (low + high) / 2 ⌋

if key == A[mid]
      return mid

else if key < A[mid]
      BinarySearch( A, key, low, mid-1 )

else if key > A[mid]
      BinarySearch( A, key, mid+1, high )
```

low = 6
high = 7
mid = (6+7)/2 = 6

key < A[6] =>
low = 6, high = 5

key > A[6] =>
low = 7, high = 7

A = { 1, 3, 4, 6, 8, 9, 11 }
key = 2

low = 0
high = 7
mid = ⌊(0+7)/2⌋ = 3

Comparison 1: key < A[3] = 6

low = 0
high = mid-1 = 3-1 = 2
mid = ⌊(0+2)/2⌋ = 1

Comparison 2: key < A[1] = 3

low = 0
high = mid-1 = 1-1 = 0
mid = ⌊(0+0)/2⌋ = 0

Comparison 3: key > A[0] = 1

low = 0
high = mid-1 = 0-1 = -1

Since (low > high) => "Not found"

A = { 1, 3, 4, 6, 8, 9, 11 }
key = 8

low = 0
high = 7
mid = ⌊(0+7)/2⌋ = 3

Comparison 1: key > A[3] = 6

low = mid+1 = 3+1 = 4
high = 7
mid = ⌊(4+7)/2⌋ = 5

Comparison 2: key < A[5] = 9

low = 4
high = mid-1 = 5-1 = 4
mid = ⌊(4+4)/2⌋ = 4

Comparison 3: key == A[4] = 8

Return (mid) => "Found at index 4"