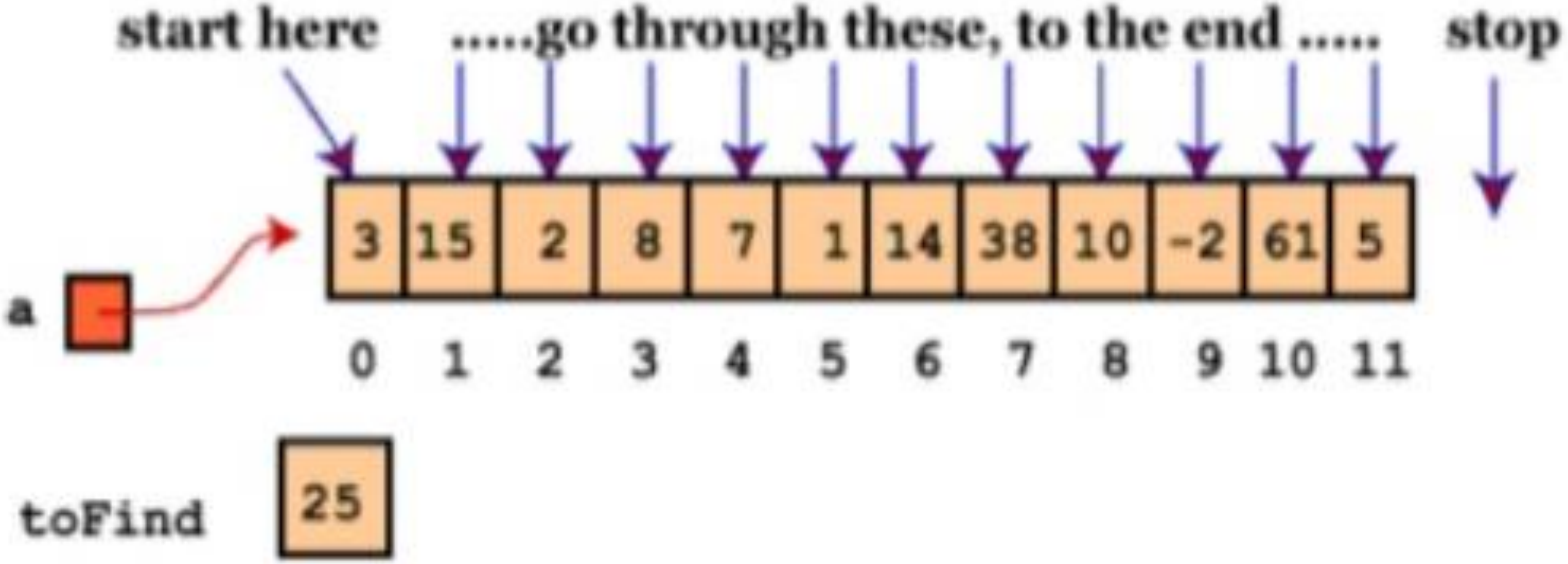
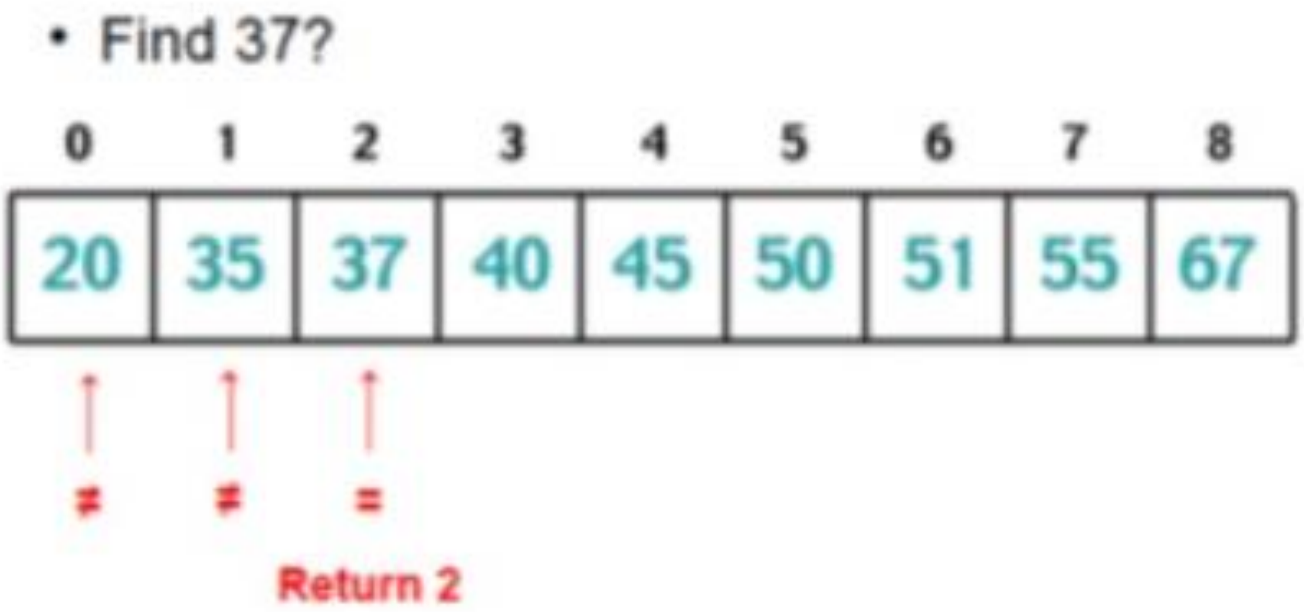
Linear Search
&
Binary Search



Every item is checked but no match is found till the end of the data collection



- Find 37?

0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67
≠	≠	=						

Return 2

Found a match at index 2

- procedure linear_search (list, value)

```
 for each item in the list
     if match item == value
                return the item's location
     end if
end for

end procedure
```

Adv. & Disadv. Of LS

- Advantages
  - Easiest to understand and implement
  - No sorting required
  - Suitable for small list sizes
  - Works fine for small number of elements

- Disadvantages
  - Time inefficient as compared to other algorithms
  - Not suitable for large-sized lists
  - Search time increases with number of elements

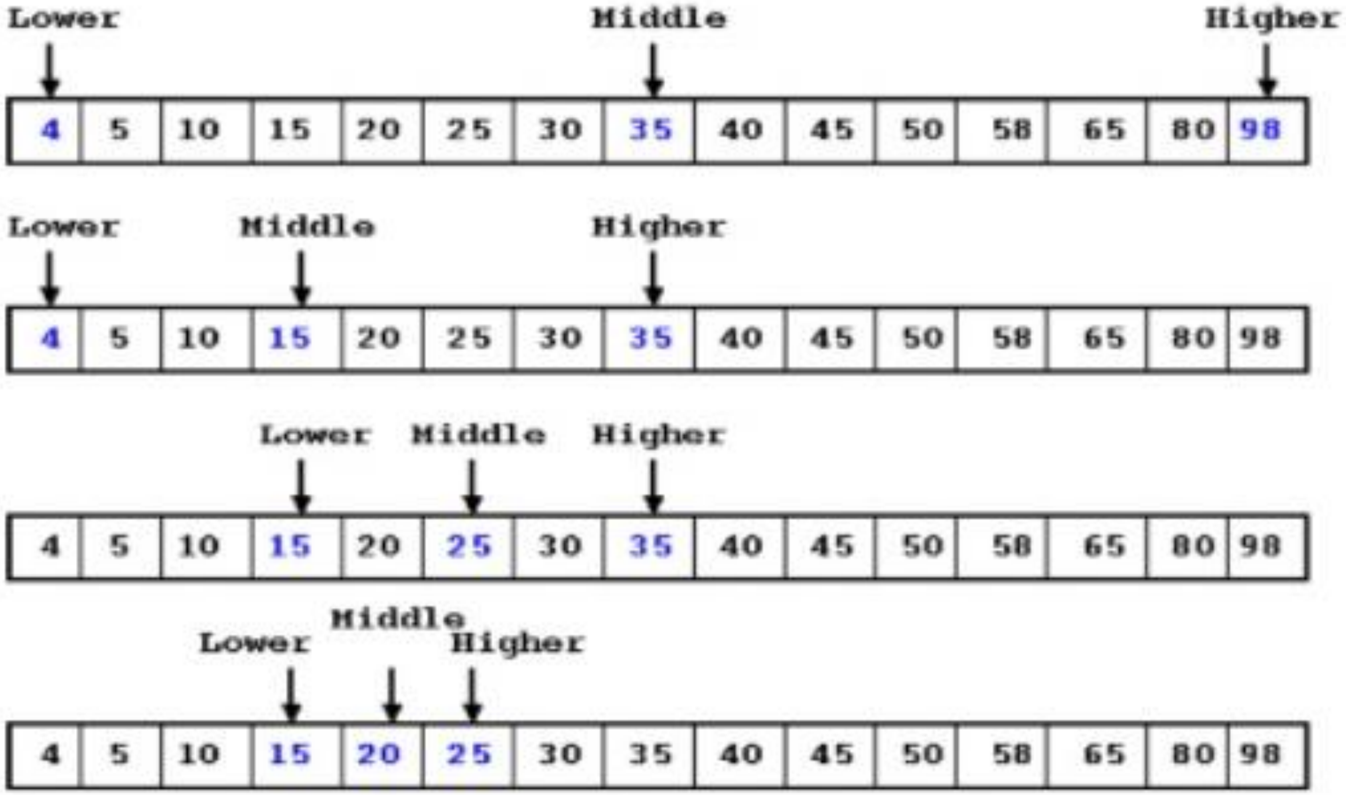
Binary Search (BS)

- Binary Search is a Divide and Conquer algorithm

- Binary search algorithm finds the position of a target value within a sorted array

- A more efficient approach than Linear Search because Binary Search basically reduces the search space to half at each step

Graphical Illustration of BS



4	5	10	15	20	25	30	35	40	45	50	58	65	80	98

4	5	10	15	20	25	30	35	40	45	50	58	65	80	98

4	5	10	15	20	25	30	35	40	45	50	58	65	80	98

4	5	10	15	20	25	30	35	40	45	50	58	65	80	98

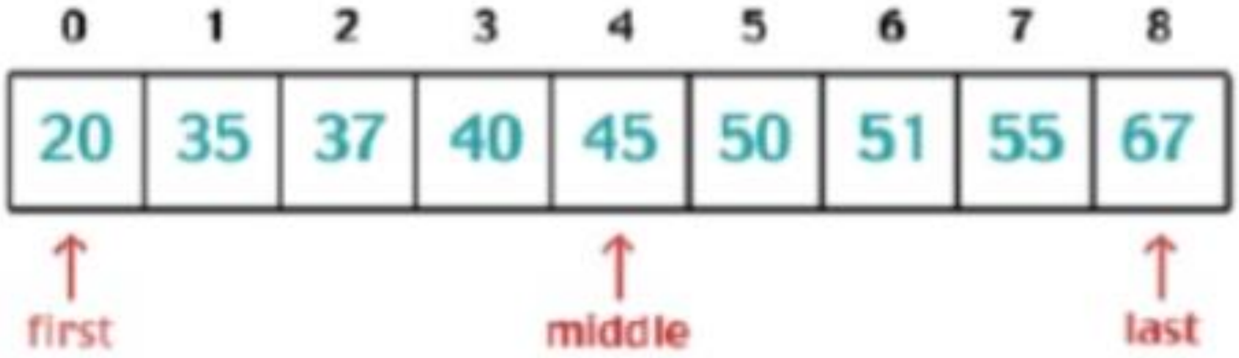
- Find 37?

1. Sort Array.

0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

Graphical Illustration of BS

2. Calculate middle = (low + high) / 2.
   = (0 + 8) / 2 = 4.



0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

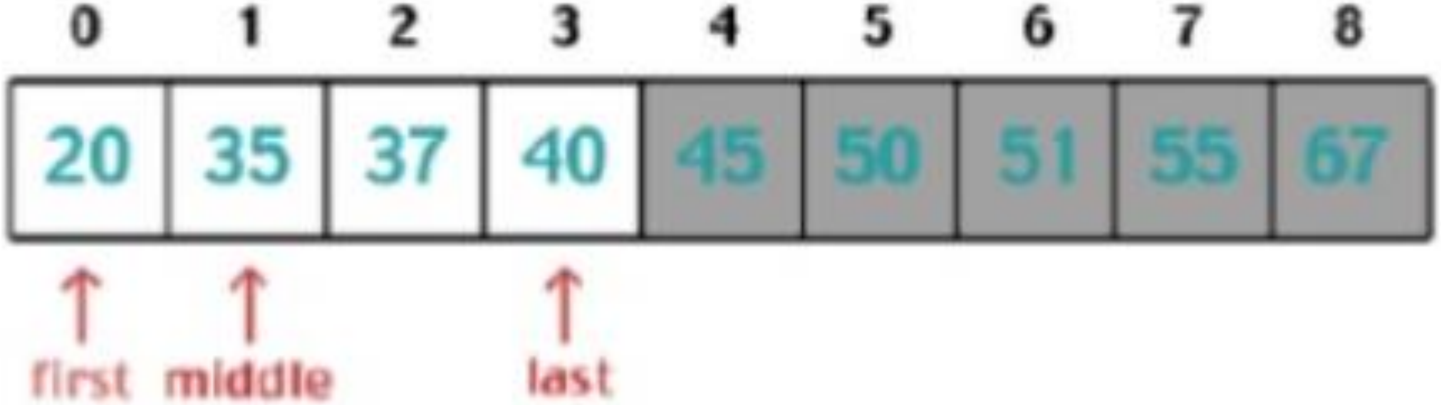
If 37 == array[middle] → return middle

Else if 37 < array[middle] → high = middle -1

Else if 37 > array[middle] → low = middle +1

Repeat 2. Calculate middle = (low + high) / 2.

= (0 + 3) / 2 = 1.



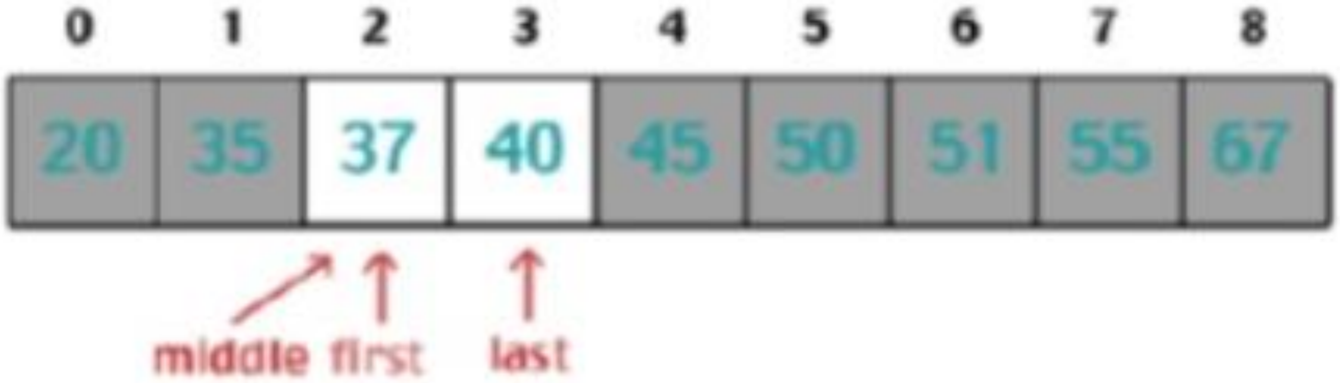
0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

If 37 == array[middle] ➔ return middle

Else if 37 < array[middle] ➔ high = middle -1

Else if 37 > array[middle] ➔ low = middle +1

Repeat 2. Calculate middle = (low + high) / 2.
= (2 + 3) / 2 = 2.



0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

If 37 == array[middle] ➔ return middle

Else if 37 < array[middle] ➔ high = middle -1

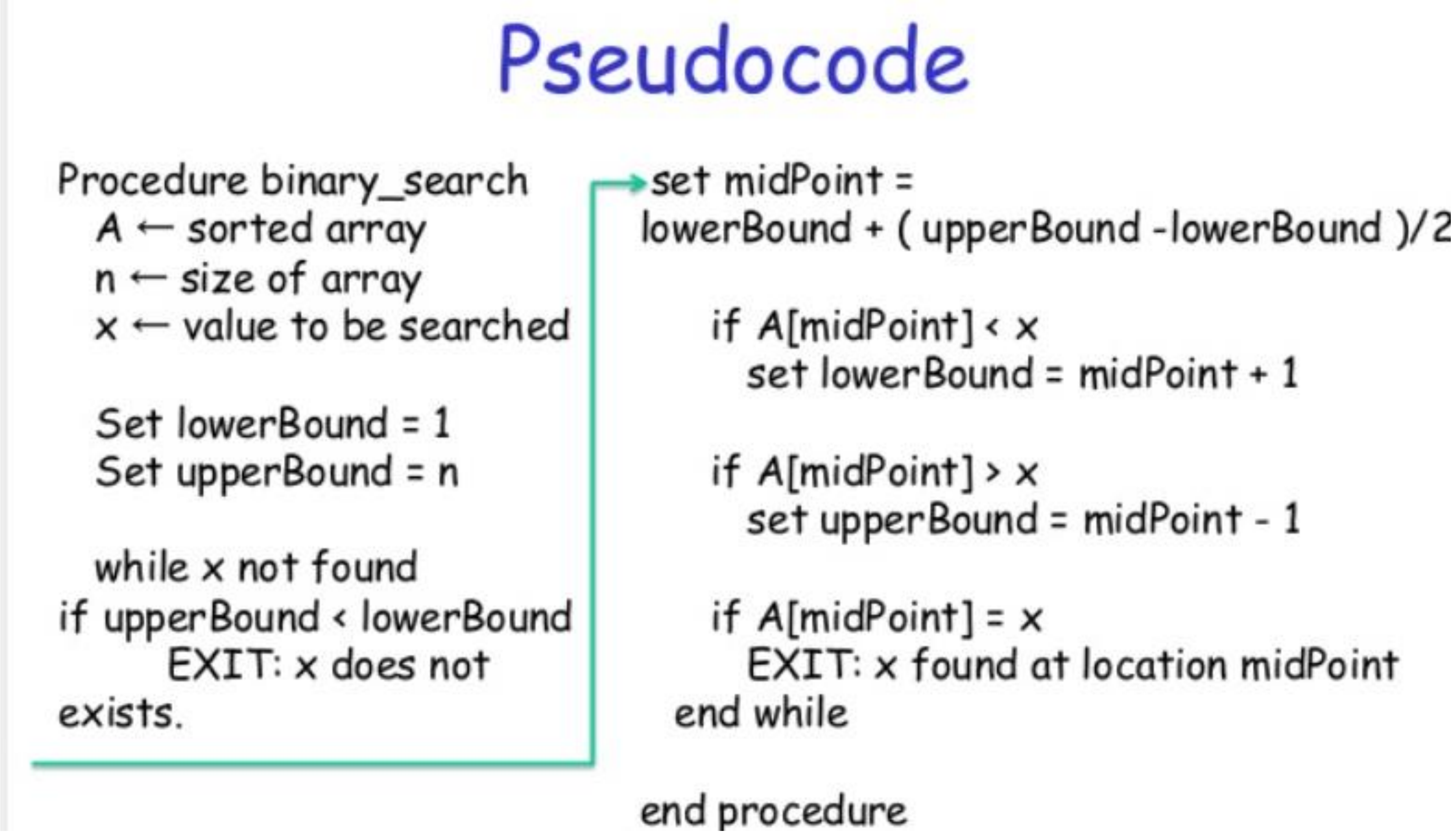
Else if 37 > array[middle] ➔ low = middle +1

Binary Search

- With each test that fails to and a match, the search is continued with one or other of the two sub-intervals, each at most half the size

- If the original number of items is N then after the first iteration there will be at most N/2 items remaining, then at most N/4 items, and so on

- In the worst case, when the value is not in the list, the algorithm must continue iterating until the list is empty



Pseudocode

```
Procedure binary_search
   A ← sorted array
   n ← size of array
   x ← value to be searched

   Set lowerBound = 1
   Set upperBound = n

   while x not found
if upperBound < lowerBound
      EXIT: x does not
exists.
set midPoint =
lowerBound + ( upperBound -lowerBound )/2

     if A[midPoint] < x
        set lowerBound = midPoint + 1

     if A[midPoint] > x
        set upperBound = midPoint - 1

     if A[midPoint] = x
        EXIT: x found at location midPoint
  end while

end procedure
```

Iterative binary search

```
int begin = 0;
int last = array.Length - 1;
int mid = 0;
```

Part #1 Initialize pointers

```
while (begin <= last) {
    mid = (begin + last) / 2;
    if (array[mid] < x) {
        begin = mid + 1;
    }
    else if (array[mid] > x) {
        last = mid - 1;
    }
    else {
        return mid;
    }
}

return -1;
```

Part #2 Search