Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	4	7	10	15	19	20	42	54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**

if $p = r$ **then**
    if $\boldsymbol{A[p] = x}$ **then** return p
    else return ***nil***
else
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    if x $\le A[q]$ **then BSearch** (A, p, q, x)
    else BSearch $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	4	7	10	15	19	20	42	54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**

if $p = r$ **then**
    if $\boldsymbol{A[p] = x}$ **then** return p
    else return ***nil***
else
    $q \leftarrow \lfloor (p + r)/2 \rfloor$
    if x $\leq A[q]$ **then BSearch** (A, p, q, x)
    else BSearch $(A, q + 1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 1, 14, 67)

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	4	7	10	15	19	20	42	54	67	75	81	87	93	96

```
BSearch(A,p,r,x):
if p = r then
    if A[p] = x then  return p
    else   return  nil
else
    q ← ⌊(p + r)/2⌋
    if x ≤ A[q] then BSearch(A,p,q,x)
    else BSearch(A,q + 1,r,x)

First call: BSearch(A,1,n,x)
```

BSearch(A, 1, 14, 67)

$q \leftarrow \lfloor (p + r)/2 \rfloor = 7$

$67 > 42 = \mathrm{A}[7]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]								54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**

if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return ***nil***
else
  $q \leftarrow \lfloor (p+r)/2 \rfloor$
  if $x \leq A[q]$ **then** **BSearch** (A, p, q, x)
  else **BSearch** $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 8, 14, 67)

$q \leftarrow \lfloor (p+r)/2 \rfloor = 11$

$67 \leq 81 = \mathrm{A}[11]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]								54	67	75	81			

```
BSearch(A,p,r,x):
if p = r then
    if A[p] = x then   return p
    else    return   nil
else
    q ← ⌊(p + r)/2⌋
    if x ≤ A[q]  then BSearch(A,p,q,x)
    else BSearch(A,q + 1,r,x)

First call: BSearch(A,1,n,x)
```

BSearch(A, 8, 11, 67)

$q \leftarrow \lfloor (p+r)/2 \rfloor = 9$

$67 \le 67 = \mathrm{A}[9]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]								54	67					

BSearch (A, p, r, x) **:**

if $p = r$ **then**
    if $\boldsymbol{A[p] = x}$ **then** return p
    else return $\boldsymbol{nil}$
else
    $q \leftarrow \lfloor (p + r)/2 \rfloor$
    if x $\leq A[q]$ **then BSearch** (A, p, q, x)
    else BSearch $(A, q + 1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 8, 9, 67)

$q \leftarrow \lfloor (p + r)/2 \rfloor = 8$

$67 > 54 = \mathrm{A}[8]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]									67					

BSearch (A, p, r, x) **:**
if $p = r$ **then**
    if $\boldsymbol{A[p] = x}$ **then** return p
    else return ***nil***
else
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    if x $\le A[q]$ **then BSearch** (A, p, q, x)
    else BSearch $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 9, 9, 67)

$p = r$ AND 67= A[p] $= A[9]$

Success

Binary search

Previous example was of a successful search, in which the search key was in the the array.

What happens if the search key is not in the array?

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	4	7	10	15	19	20	42	54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**

if $p = r$ **then**

  if $A[p] = x$ **then** return p

  else return ***nil***

else

  $q \leftarrow \lfloor (p + r)/2 \rfloor$

  if $x \leq A[q]$ **then** **BSearch** (A, p, q, x)

  else **BSearch** $(A, q + 1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 1, 14, 66)

Binary search

Input: An array A of elements in sorted order, and an element x.

Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	4	7	10	15	19	20	42	54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**

if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return ***nil***
else
  $q \leftarrow \lfloor (p + r)/2 \rfloor$
  if $x \leq A[q]$ **then** **BSearch** (A, p, q, x)
  else **BSearch** $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 1, 14, 66)

$q \leftarrow \lfloor (p + r)/2 \rfloor = 7$

$66 > 42 = \mathrm{A}[7]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]								54	67	75	81	87	93	96

BSearch (A, p, r, x) **:**
if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return ***nil***
else
  $q \leftarrow \lfloor (p+r)/2 \rfloor$
  if x $\le A[q]$ **then** **BSearch** (A, p, q, x)
  else **BSearch** $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 8, 14, 66)

$q \leftarrow \lfloor (p+r)/2 \rfloor = 11$

$66 \le 81 = \mathrm{A}[11]$

Binary search

Input: An array A of elements in sorted order, and an element x.

Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]								54	67	75	81			

BSearch (A, p, r, x) **:**

if $p = r$ **then**

  if $A[p] = x$ **then** return p

  else return ***nil***

else

  $q \leftarrow \lfloor (p + r)/2 \rfloor$

  if $x \leq A[q]$ **then** **BSearch** (A, p, q, x)

  else **BSearch** $(A, q + 1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 8, 11, 66)

$q \leftarrow \lfloor (p + r)/2 \rfloor = 9$

$66 \leq 67 = \mathrm{A}[9]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \leq A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	54	67	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

BSearch (A, p, r, x) **:**

if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return ***nil***
else
  $q \leftarrow \lfloor (p+r)/2 \rfloor$
  if $x \leq A[q]$ **then** **BSearch** (A, p, q, x)
  else **BSearch** $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

BSearch(A, 8, 9, 66)

$q \leftarrow \lfloor (p+r)/2 \rfloor = 8$

$67 > 54 = \mathrm{A}[8]$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]									67					

BSearch (A, p, r, x) **:**

if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return nil
else
  $q \leftarrow \lfloor (p+r)/2 \rfloor$
  if $x \le A[q]$ **then** BSearch (A, p, q, x)
  else BSearch $(A, q+1, r, x)$

First call: BSearch $(A, 1, n, x)$

BSearch(A, 9, 9, 66)

$p = r$ AND $66 \neq A[p] = A[9]$

Failure
Return $\boldsymbol{nil}$

Binary search

Input: An array A of elements in sorted order, and an element x.
Output: Return the position of x if it exists; otherwise output nil.

Idea: Set q = middle item. Check if $x \le A[q]$,
and search either to left or right of q as appropriate

q =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A[q]									67					

BSearch(A, 9, 9, 66)

Failure

Return *nil*

In fact, algorithm knows where 66 *should* be. In case of failure it could be modified to return locations of predecessor and successor of x in the array.

BSearch (A, p, r, x) **:**

if $p = r$ **then**
  if $A[p] = x$ **then** return p
  else return nil
else
  $q \leftarrow \lfloor (p+r)/2 \rfloor$
  if $x \le A[q]$ **then** **BSearch** (A, p, q, x)
  else **BSearch** $(A, q+1, r, x)$

First call: **BSearch** $(A, 1, n, x)$

Analysis of Binary Search

Analysis: Let $T(n)$ be the number of comparisons needed for running the algorithm on at most n elements. (Assume n a power of 2).

Recurrence: A single comparison eliminates half of the array. Therefore, we search for the element in the remaining half, which has size $n/2$. Thus, the recurrence counting the number of comparisons is:

$$T(1) = 1. \textit{ Otherwise, if } n > 1,\ T(n) = T\left(\frac{n}{2}\right) + 1.$$

Solve the recurrence by iterating it: *(Assume n a power of 2)*

$$\begin{aligned}
T(n) &= T(n/2) \quad +1 \\
&= (T(n/2^2)+1) +1 \\
&= T(n/2^2) \quad +2 \\
&= \ldots \\
&= T(n/2^i) \quad +i \\
&= \ldots \\
&= T(1)+\log n = \Theta(\log n)
\end{aligned}$$

Proof of correctness
more formal analysis and analysis when n is not a power of 2 are developed in practice homework

Note: may sometimes terminate faster than $\Theta(\log n)$, but **worst-case running time** is $\Theta(\log n)$.