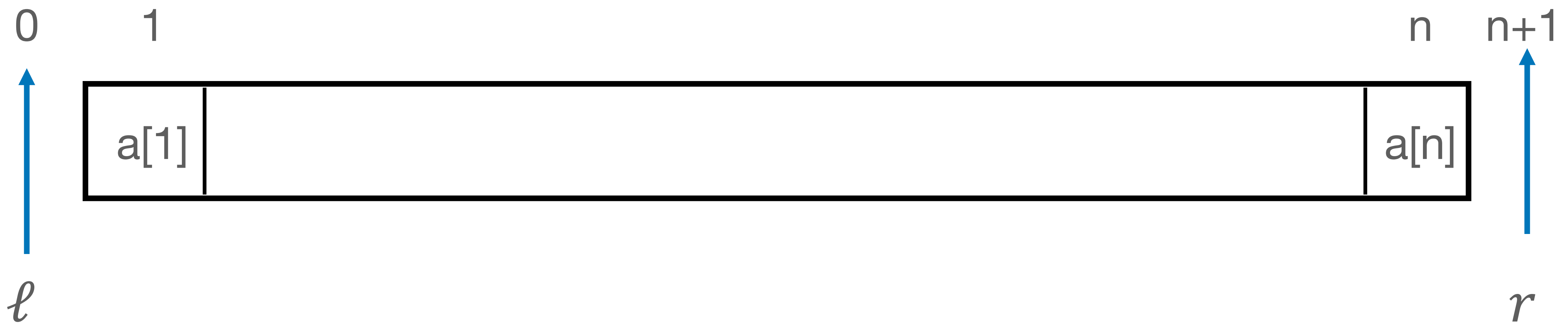
Binary Search

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

```
ℓ := 0; r := n + 1;
while ℓ + 1 < r do   /* 0 ≤ ℓ < r ≤ n + 1 AND a[ℓ] < x < a[r] */
    m := ⌊(ℓ+r)/2⌋;
    if a[m] = x then
        return m;
    if a[m] < x then
        ℓ := m
    else
        r := m;
return −1;
```

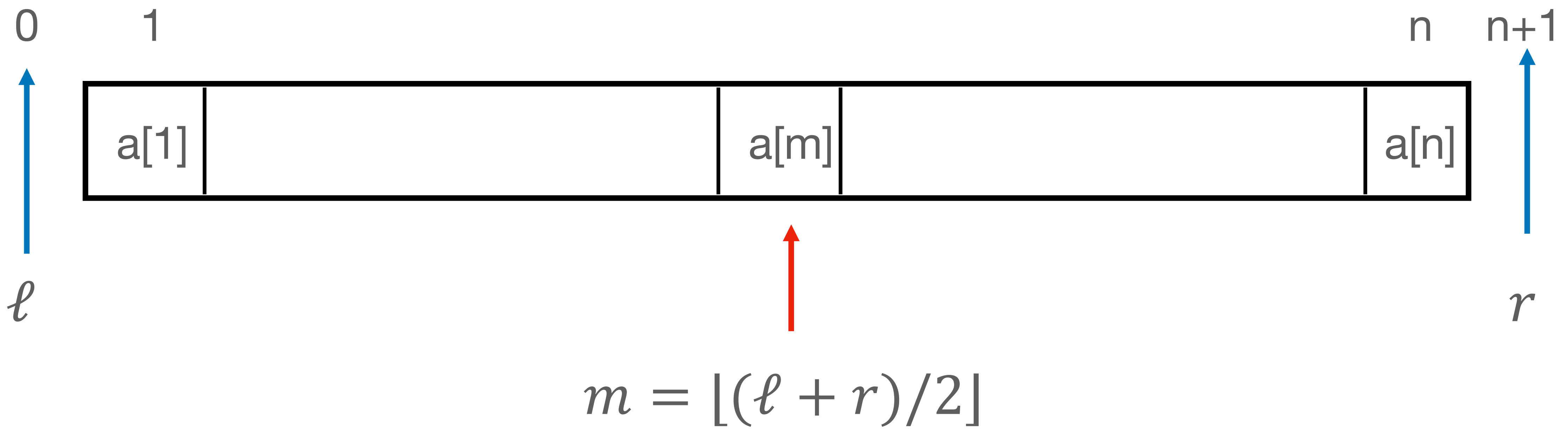


Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n + 1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n + 1$ AND $a[\ell] < x < a[r]$ */
  $m := \lfloor \frac{\ell + r}{2} \rfloor$;
  if $a[m] = x$ **then**
    return m;
  if $a[m] < x$ **then**
    $\ell := m$
  else
    $r := m$;
return -1;

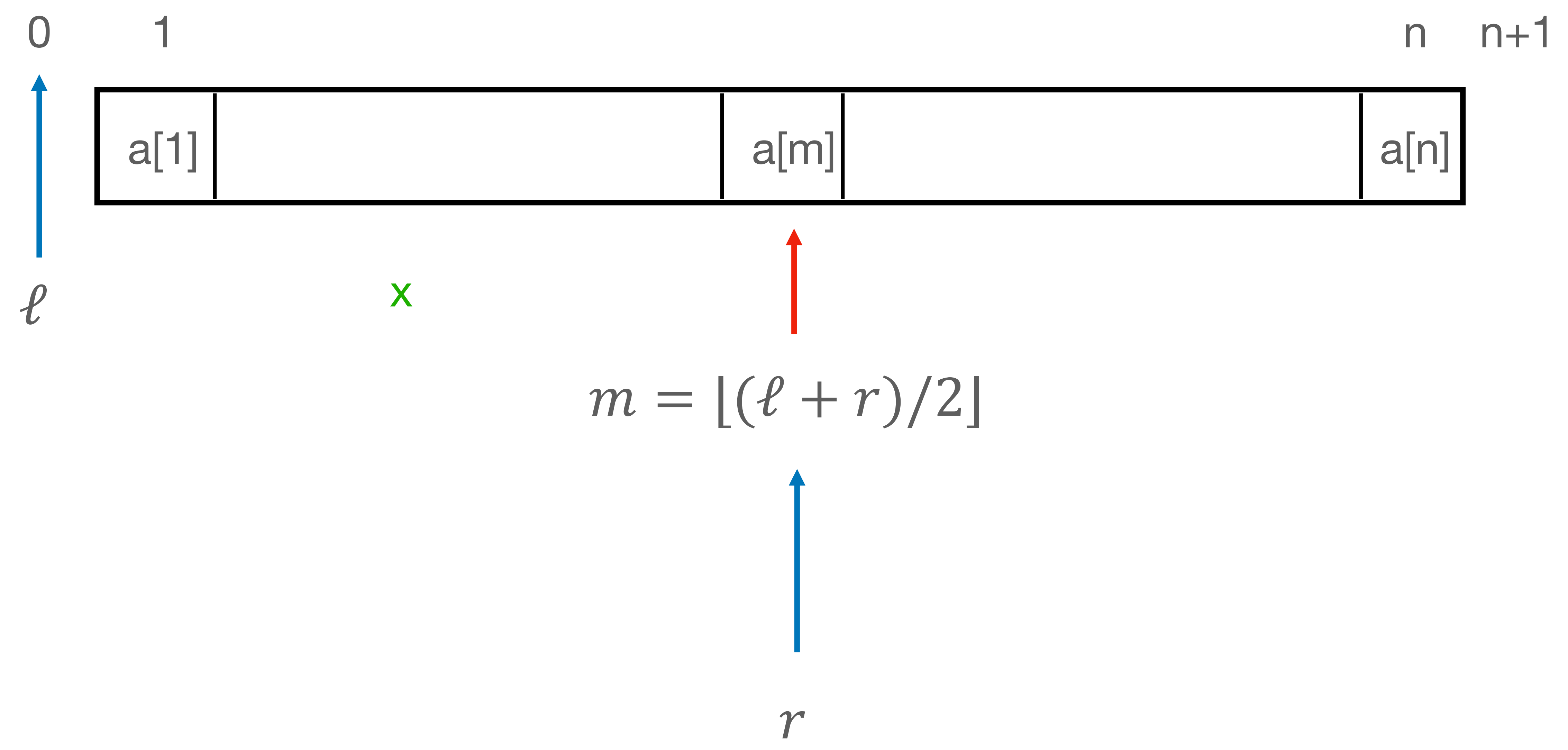


Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \leq m \leq n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n+1$;
while $\ell + 1 < r$ **do** /* $0 \leq \ell < r \leq n+1$ AND $a[\ell] < x < a[r]$ */
  $m := \lfloor \frac{\ell+r}{2} \rfloor$;
  if $a[m] = x$ **then**
    return m;
  if $a[m] < x$ **then**
    $\ell := m$
  else
    $r := m$;
return -1;

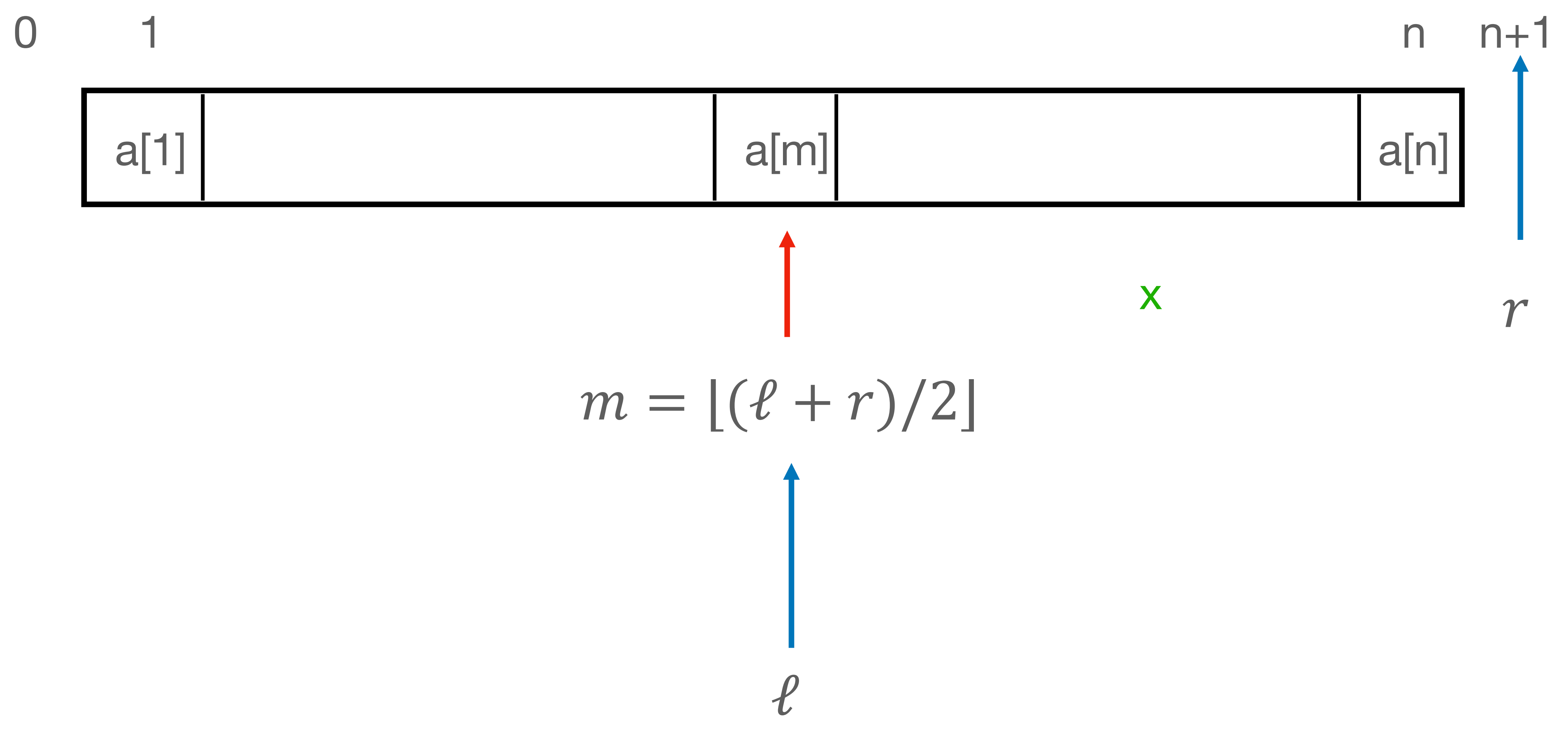


Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n + 1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n + 1$ AND $a[\ell] < x < a[r]$ */
  $m := \lfloor \frac{\ell + r}{2} \rfloor$;
  if $a[m] = x$ **then**
    return m;
  if $a[m] < x$ **then**
    $\ell := m$
  else
    $r := m$;
return -1;

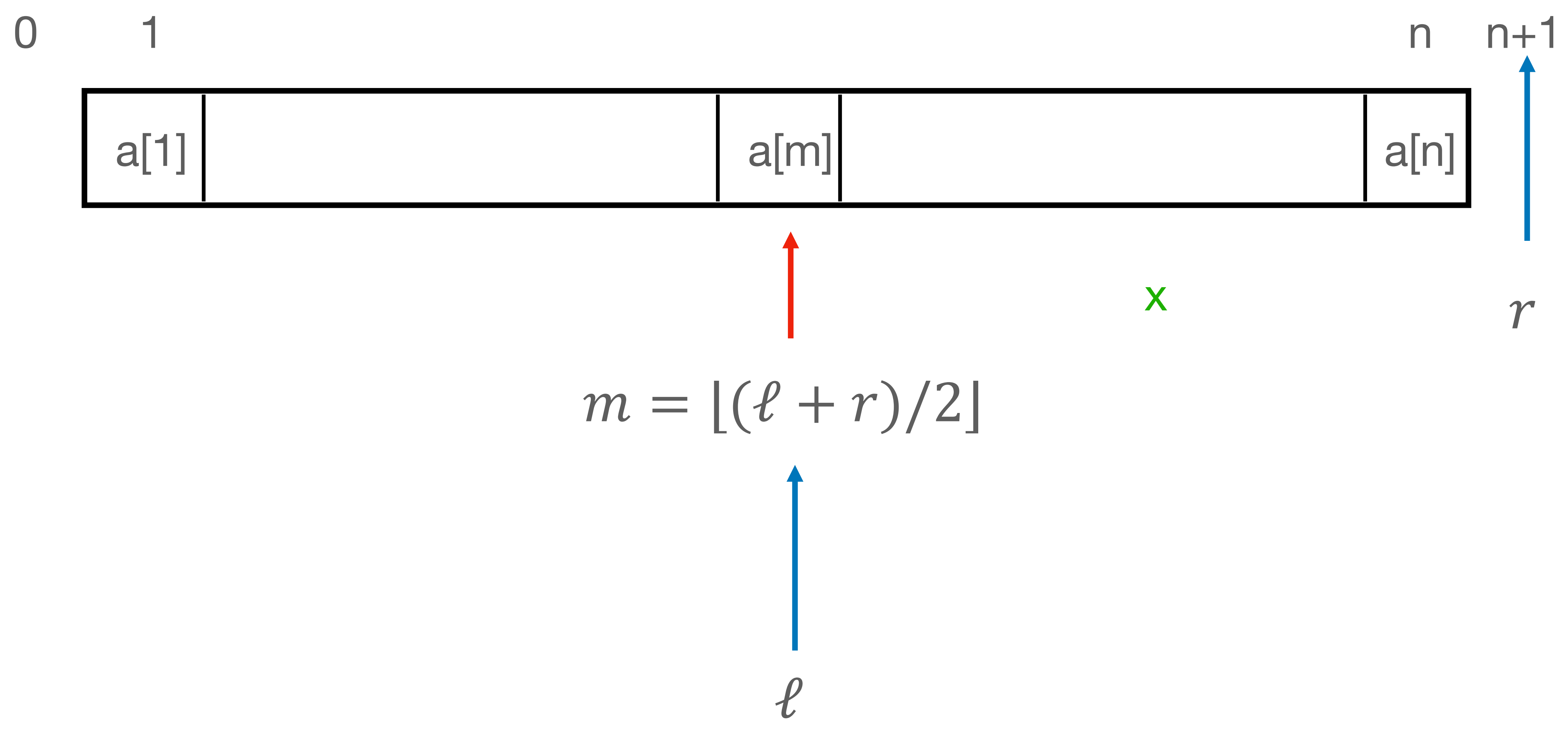


Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n+1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n+1$ AND $a[\ell] < x < a[r]$ */
    $m := \lfloor \frac{\ell+r}{2} \rfloor$;
    if $a[m] = x$ **then**
        return m;
    if $a[m] < x$ **then**
        $\ell := m$
    else
        $r := m$;
return -1;

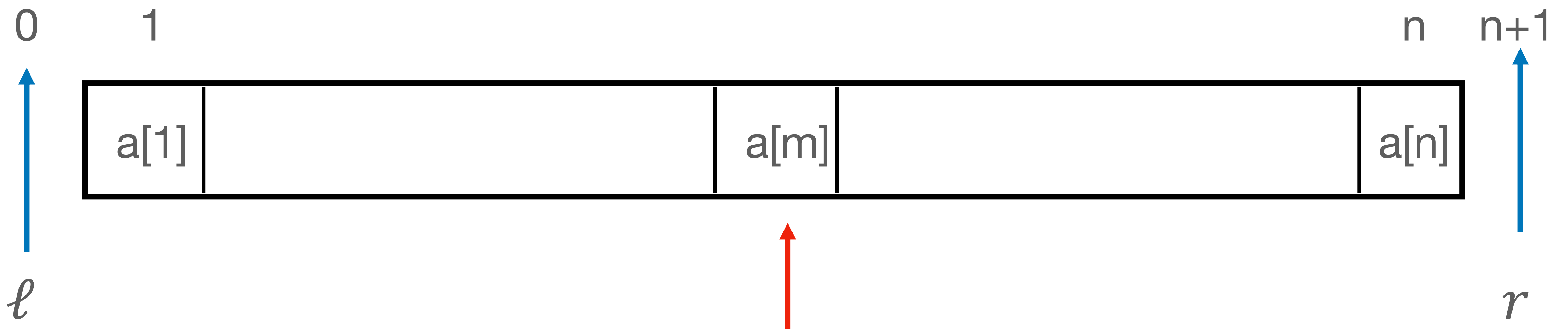


Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n + 1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n+1$ AND $a[\ell] < x < a[r]$ */
    $m := \lfloor \frac{\ell + r}{2} \rfloor$;
    if $a[m] = x$ **then**
        return m;
    if $a[m] < x$ **then**
        $\ell := m$
    else
        $r := m$;
return -1;



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
  - almost due to Gauss brackets

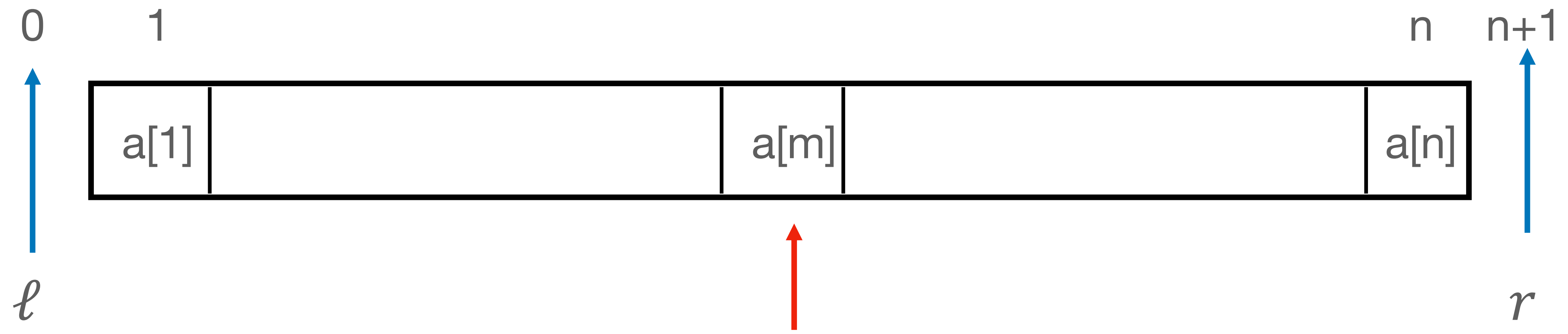
trick: problem size $n = r - \ell - 1$

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n+1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n+1$ AND $a[\ell] < x < a[r]$ */
  $m := \lfloor \frac{\ell+r}{2} \rfloor$;
  if $a[m] = x$ **then**
    return m;
  if $a[m] < x$ **then**
    $\ell := m$
  else
    $r := m$;
return -1;



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
  - almost due to Gauss brackets

trick: problem size $\quad n = r - \ell - 1$

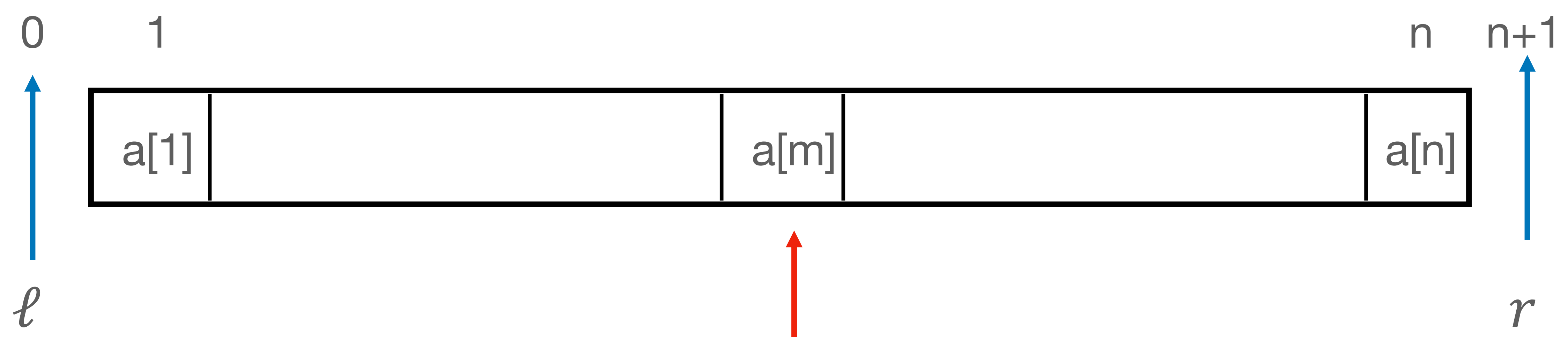
after next pass of loop $\quad n' = r' - \ell' - 1$

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

$\ell := 0$; $r := n + 1$;
while $\ell + 1 < r$ **do** /* $0 \le \ell < r \le n+1$ AND $a[\ell] < x < a[r]$ */
  $m := \lfloor \frac{\ell + r}{2} \rfloor$;
  if $a[m] = x$ **then**
    return m;
  if $a[m] < x$ **then**
    $\ell := m$
  else
    $r := m$;
return -1;



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
  - almost due to Gauss brackets

trick: problem size $n = r - \ell - 1$

after next pass of loop $n' = r' - \ell' - 1$

Lemma 1.

$$n' \le n/2$$

Proof.

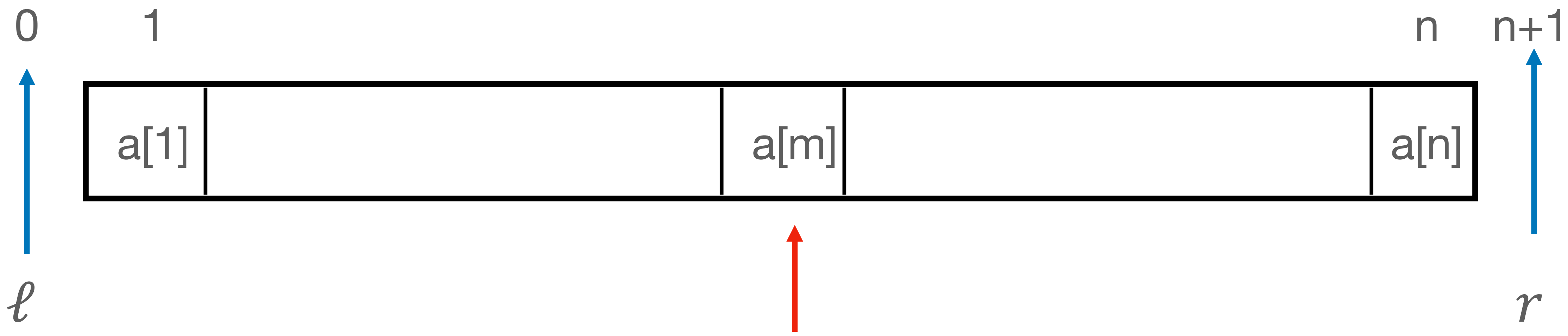
$$(r+\ell)/2 - 1/2 \le m = \lfloor (r+\ell)/2) \rfloor \le (r + \ell/2)$$

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

```
ℓ := 0; r := n + 1;
while ℓ + 1 < r do    /* 0 ≤ ℓ < r ≤ n + 1 AND a[ℓ] < x < a[r] */
    m := ⌊(ℓ+r)/2⌋;
    if a[m] = x then
        return m;
    if a[m] < x then
        ℓ := m
    else
        r := m;
return −1;
```



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
  - almost due to Gauss brackets

trick: problem size $\quad n = r - \ell - 1$

after next pass of loop $\quad n' = r' - \ell' - 1$

Lemma 1.

$$n' \le n/2$$

Proof.

$$(r+\ell)/2 - 1/2 \le m = \lfloor (r+\ell)/2) \rfloor \le (r + \ell/2)$$

- $x < m$:

$$\ell' = \ell\,,\ r' = m$$

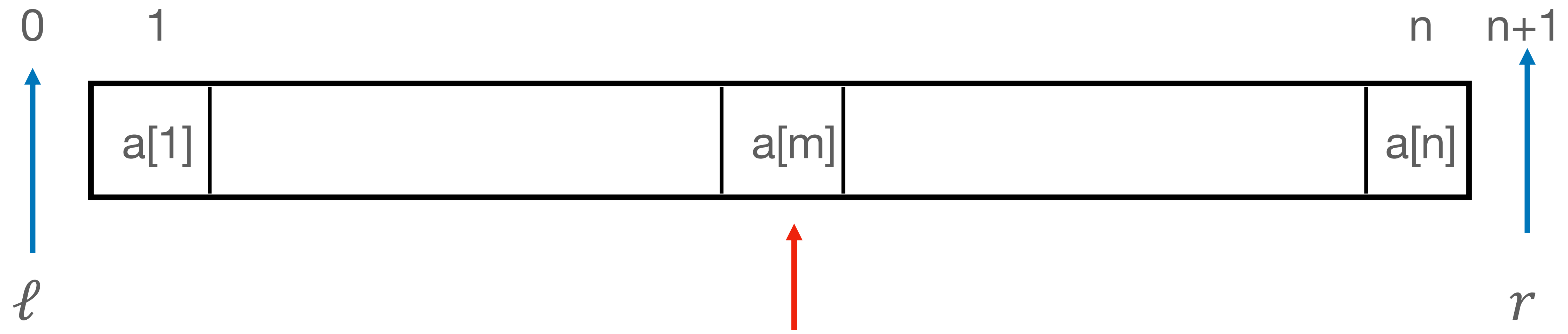
$$\begin{aligned} n' &= m - \ell - 1 \\ &= \lfloor (r+\ell)/2 \rfloor - \ell - 1 \\ &\le (r+\ell)/2 - \ell - 1/2 \\ &= (r+\ell-1)/2 \end{aligned}$$

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

```
ℓ := 0; r := n + 1;
while ℓ + 1 < r do   /* 0 ≤ ℓ < r ≤ n + 1 AND a[ℓ] < x < a[r] */
    m := ⌊(ℓ+r)/2⌋;
    if a[m] = x then
        return m;
    if a[m] < x then
        ℓ := m
    else
        r := m;
return −1;
```



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
    - almost due to Gauss brackets

trick: problem size $\quad n = r - \ell - 1$

after next pass of loop $\quad n' = r' - \ell' - 1$

Lemma 1.

$$n' \le n/2$$

Proof.

$$(r+\ell)/2 - 1/2 \le m = \lfloor (r+\ell)/2) \rfloor \le (r + \ell/2)$$

- $x > m$:

$$\ell' = m\ ,\ r' = r$$

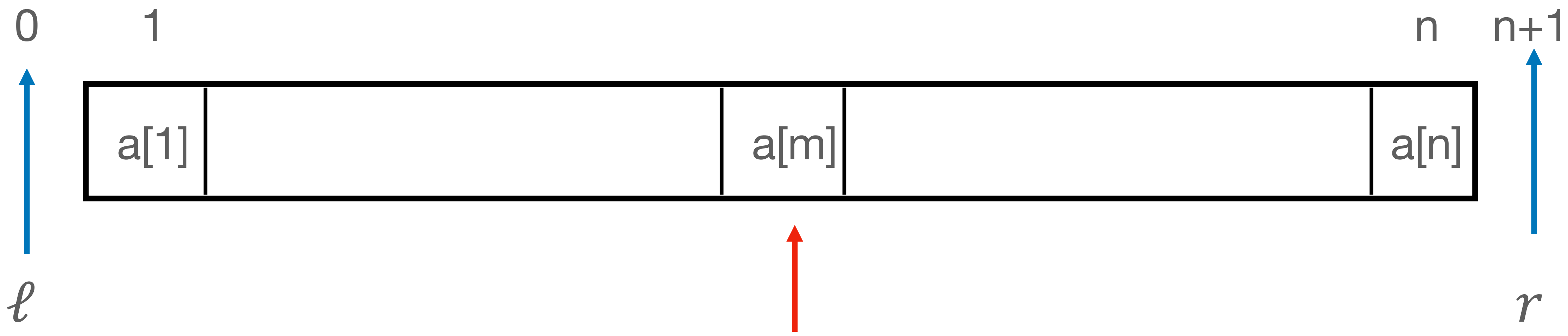
$$\begin{aligned} n' &= r - m - 1 \\ &= r - \lfloor (r+\ell)/2 \rfloor - 1 \\ &\le r - ((r+\ell)/2 - 1/2) - 1 \\ &= (r + \ell - 1)/2 \end{aligned}$$

Algorithm 1 Binary-search

Input: Sorted array $a[1..n]$, $a[1] < a[2] < \cdots < a[n]$, element x

Output: $\begin{cases} m & \text{if there is an } 1 \le m \le n \text{ with } a[m] = x \\ -1 & \text{otherwise} \end{cases}$

```
ℓ := 0; r := n + 1;
while ℓ + 1 < r do    /* 0 ≤ ℓ < r ≤ n + 1 AND a[ℓ] < x < a[r] */
    m := ⌊(ℓ+r)/2⌋;
    if a[m] = x then
        return m;
    if a[m] < x then
        ℓ := m
    else
        r := m;
return −1;
```



$$m = \lfloor (\ell + r)/2 \rfloor$$

- problem size halving with each comparison
    - almost due to Gauss brackets

trick: problem size $\quad n = r - \ell - 1$

after next pass of loop $\quad n' = r' - \ell' - 1$

T(n): number of comparisons

$$T(1) = 0\,, T(n) \le \boxed{2} + T(n/2)$$

$$\begin{aligned} T(n) &\le 2 + T(n/2) \\ &\le 4 + T(n/2^2) \\ &= 2x + T(n/2^x) \\ x &= \log n \\ T(n) &\le 2\log n \end{aligned}$$