Algorithms & Data Structures I
CSC 225

Ali Mashreghi

Fall 2018

Department of Computer Science, University of Victoria

Searching

Searching Problem

Input: Given an input array $A[1..n]$, and a number x

Output: If x is in A, return an index i such that $A[i] = x$.
Otherwise, return -1.

Ex. input: $\mathrm{A} = \{7, 3, 2, 5, 2, 8, 2, 9\}, x = 2$ output: 5

Ex. input: $\mathrm{A} = \{7, 3, 12, 5, 6, 8, 2, 9\}, x = 4$ output: -1

Searching

SEARCHING PROBLEM

Input: Given an input array $A[1..n]$, and a number x

Output: If x is in A, return an index i such that $A[i] = x$. Otherwise, return -1.

Ex. input: $\mathrm{A} = \{7, 3, 2, 5, 2, 8, 2, 9\}, x = 2$ output: 5

Ex. input: $\mathrm{A} = \{7, 3, 12, 5, 6, 8, 2, 9\}, x = 4$ output: -1

- In case there are multiple answers any of them is acceptable.

Searching

- Trivial solution is $O(n)$, and that's the best we can do if the array is not sorted.
- Things get interesting when we considered a variation in which the input array is sorted.

Searching in a sorted array

Input: Given a sorted array $A[1..n]$, and a number x

Output: If x is in A, return an index i such that $A[i] = x$. Otherwise, return -1.

Binary Search

Search for $x = 28$

1 16

2	**3**	**5**	**7**	**11**	**13**	**14**	**17**	**23**	**28**	**31**	**39**	**55**	**90**	**97**	**99**

low = 1 *high = 16*

Binary Search

Search for $x = 28$

1 $\quad$ 8 $\quad$ 16

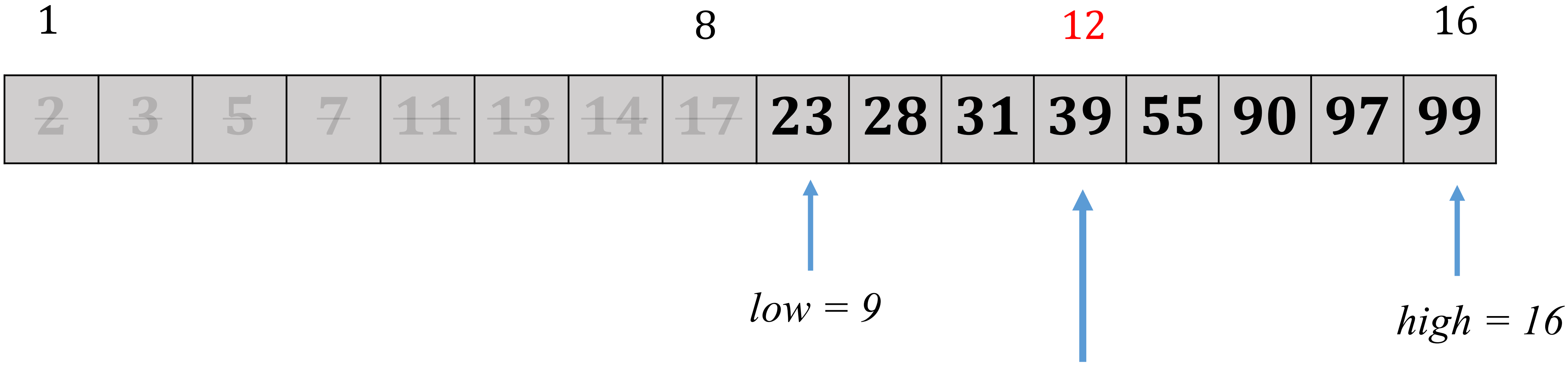
2	**3**	**5**	**7**	**11**	**13**	**14**	**17**	**23**	**28**	**31**	**39**	**55**	**90**	**97**	**99**

low = 1 $\quad$ *mid = (low + high)/2* $\quad$ *high = 16*

1) Compare $x = 28$ with $A[8]$
2) If $x == A[8]$, done
   If $x > A[8]$, search the right side
   If $x < A[8]$, search the left side

Binary Search

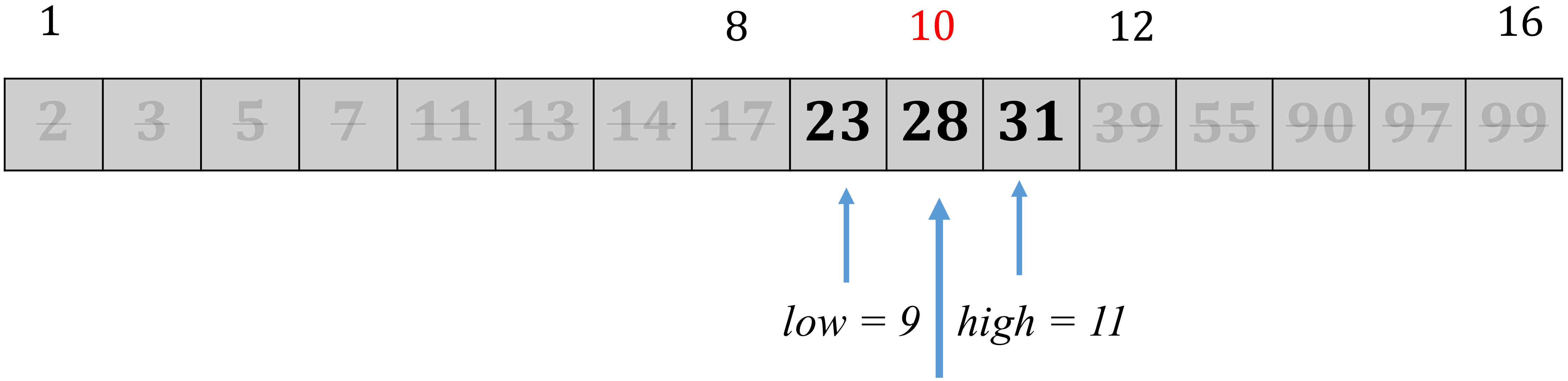
Search for $x = 28$



Compare $x = 28$ with $A[12]$

Binary Search

Search for $x = 28$



Compare $x = 28$ with $A[10]$

Binary Search

Search for $x = 28$

1							8		10		12				16
~~2~~	~~3~~	~~5~~	~~7~~	~~11~~	~~13~~	~~14~~	~~17~~	**23**	**28**	**31**	~~39~~	~~55~~	~~90~~	~~97~~	~~99~~

Return 10 as the answer

```
BINARY-SEARCH(A, x)
    low = 1 // beginning of search range
    high = n+1 // end of search range
    while low < high
        mid = ⌊(low+high)/2⌋
        if x == A[mid]
            return mid
        elseif x > A[mid]
            low = mid + 1
        else
            high = mid
    return -1
```

```
BINARY-SEARCH(A, x)
    low = 1 // beginning of search range
    high = n+1 // end of search range
    while low < high
        mid = ⌊(low+high)/2⌋
        if x == A[mid]
            return mid
        elseif x > A[mid]
            low = mid + 1
        else
            high = mid
    return -1
```

- This implementation is a bit different from the slides.
- We consider that high **is excluded** from the search range.
- The reason is that usually in programming languages the end index is exclusive. For example, Java's substring method.

BINARY-SEARCH(A, x)

```
1  low = 1 // beginning of search range
2  high = n+1 // end of search range
3  while low < high
4      mid = ⌊(low+high)/2⌋
5      if x == A[mid]
6          return mid
7      elseif x > A[mid]
8          low = mid + 1
9      else
10         high = mid
11 return -1
```

Question: What happens if we change line 3 to $low \leq high$?

```
BINARY-SEARCH(A, x)
1  low = 1 // beginning of search range
2  high = n+1 // end of search range
3  while low < high
4      mid = ⌊(low+high)/2⌋
5      if x == A[mid]
6          return mid
7      elseif x > A[mid]
8          low = mid + 1
9      else
10         high = mid
11 return -1
```

Question: What happens if we change line 3 to $low \leq high$?

Answer: It could result in an infinite loop. For example, when $A = \{3\}$, and $x = 2$

Analysis

- In each iteration, either the answer is found or the search range is **divided by 2**.

- In each iteration of the while loop we do **constant work**

- Therefore, $T(n) = O(\log n)$

Recursive implementation

```
BINARY-SEARCH(A, low, high, x)
    mid = ⌊(low+high)/2⌋
    if x == A[mid]
        return mid
    elseif x > A[mid]
        return BINARY-SEARCH(A, mid + 1, high, x)
    else
         return BINARY-SEARCH(A, low, mid, x)
    return -1
```

Recursive implementation

- The recurrence is $T(n) \leq T\left(\frac{n}{2}\right) + \Theta(1)$
- So, in the worst-case $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$
- Using Master theorem (case 2), $T(n) = \boldsymbol{\Theta(\log n)}$

Recursive implementation

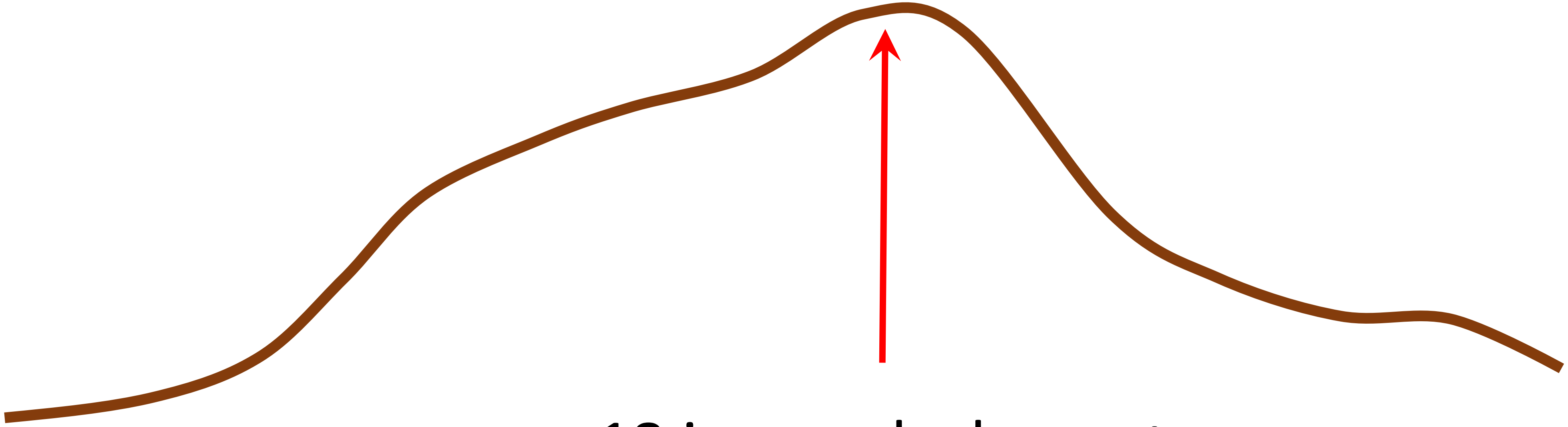
- The recurrence is $T(n) \leq T\left(\frac{n}{2}\right) + \Theta(1)$
- So, in the worst-case $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$
- Using Master theorem (case 2), $T(n) = \boldsymbol{\Theta(\log n)}$
- The actual running time is $\boldsymbol{O(\log n)}$ because of the $\leq$

An interesting problem

-9	-8	-5	0	7	9	11	13	18	17	9	3	2	2	0

Peak finding

-9	-8	-5	0	7	9	11	13	18	17	9	3	2	2	0



18 is a peak element

Formal definition

Peak Finding

Input: An array $A[1..n]$ that is guaranteed to have some index i $(1 \leq i \leq n)$, such that $A[1] \leq \cdots \leq A[i]$, and $A[i] \geq \cdots \geq A[n]$

Output: Find any index j such that $A[j-1] \leq A[j] \geq A[j+1]$

If $j = \ 1$ only $\leq$ should hold
If $j = n$ only $\geq$ should hold

Basic idea

1							8							15
-9	**-8**	**-5**	**0**	**7**	**9**	**11**	**13**	**18**	**17**	**9**	**3**	**2**	**2**	**0**

Question: What is the condition that I should check at the middle index?

Basic idea

1							8							15
−9	**−8**	**−5**	**0**	**7**	**9**	**11**	**13**	**18**	**17**	**9**	**3**	**2**	**2**	**0**

Answer: Look at the middle element in the search range, if the numbers are increasing, like $A[mid\ -1\ ] \le A[mid] \le A[mid+1]$, go right. Otherwise, go left.