

# **Algoritmer og Datastrukturer**

Binære søgetræer  
[CLRS, kapitel 12]

# Abstrakt Datastruktur: Ordbog

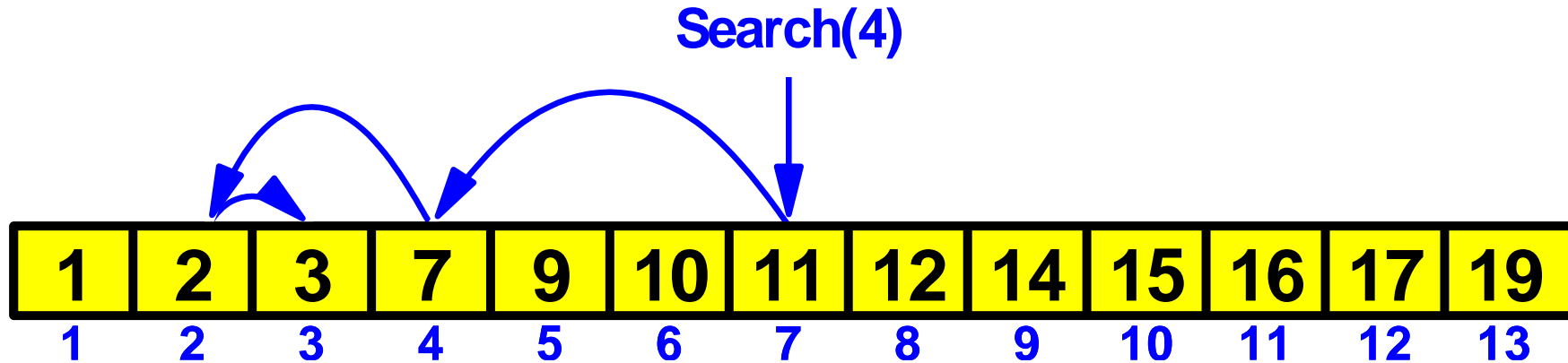
**Search( $S, x$ )**

**Insert( $S, x$ )**

**Delete( $S, x$ )**

Elementer må kun sammenlignes med " $\leq$ "

# (Statisk) Ordbog : Sorteret Array



<b>Search(<math>S, x</math>)</b>	$O(\log n)$
<b>Insert(<math>S, x</math>)</b>	$O(n)$
<b>Delete(<math>S, x</math>)</b>	

# Worst-case Søgetid

Hvor mange sammenligninger kræver det at søge efter et element i et sorteret array med 8 elementer, når man kun kan lave sammenligninger ?



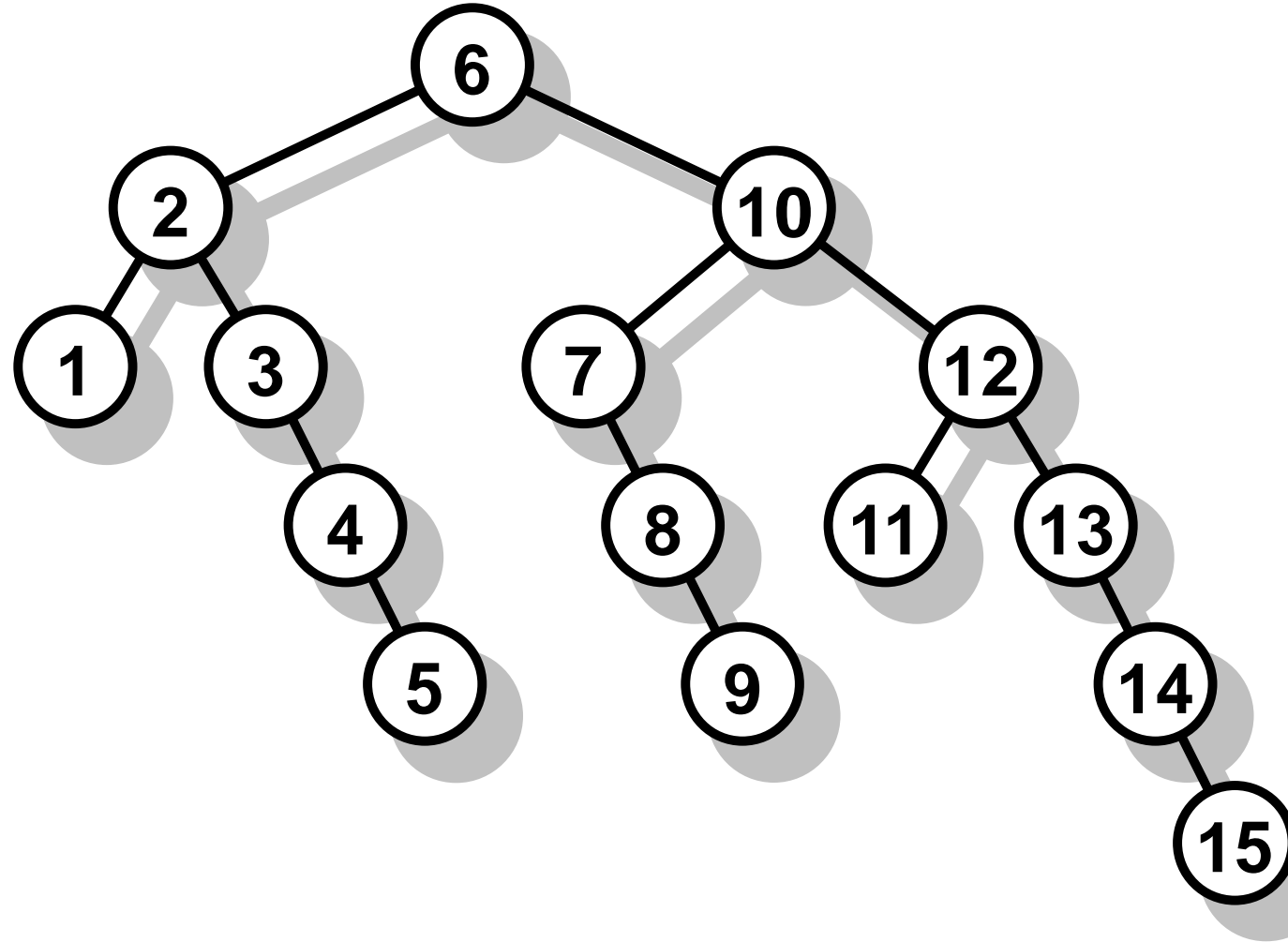
- a) 1
- b) 2
- c) 3
- d) 4
- e) 5
- f) 6
- g) 7
- h) 8
- i) Ved ikke

1	2	3	4	5	6	7	8
2	5	7	9	11	13	42	71

## Sætning

At søge blandt  $\geq 2^i - 1$  elementer kræver mindst  $i$  sammenligninger

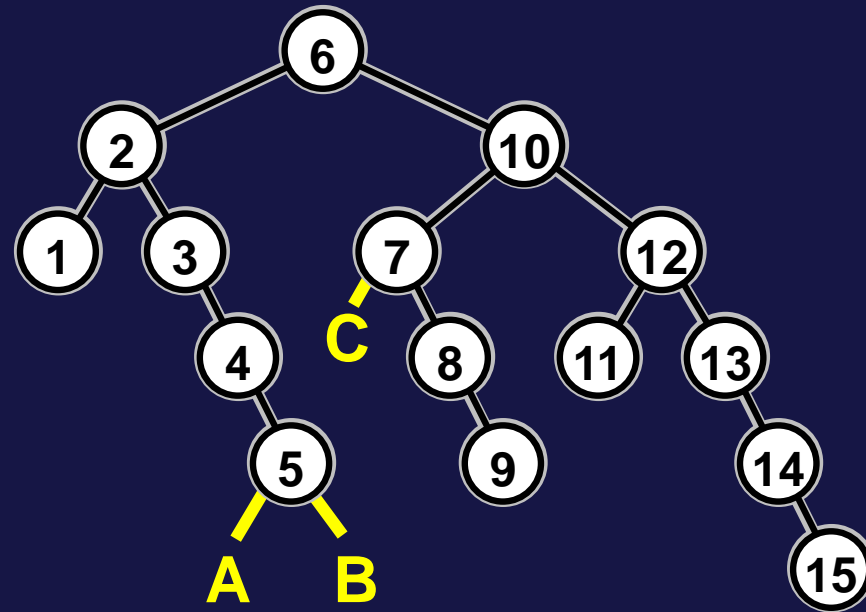
# Søgetræ



**Invariant** For alle knuder er elementerne i venstre (højre) undertræ mindre (større) end eller lig med elementet i knuden

# Hvor kan 5.5 indsættes ?

- a) A
- 😊 b) B
- c) C
- d) A eller B
- e) B eller C
- f) ved ikke



# Søgetræs søgninger

TREE-SEARCH( $x, k$ )

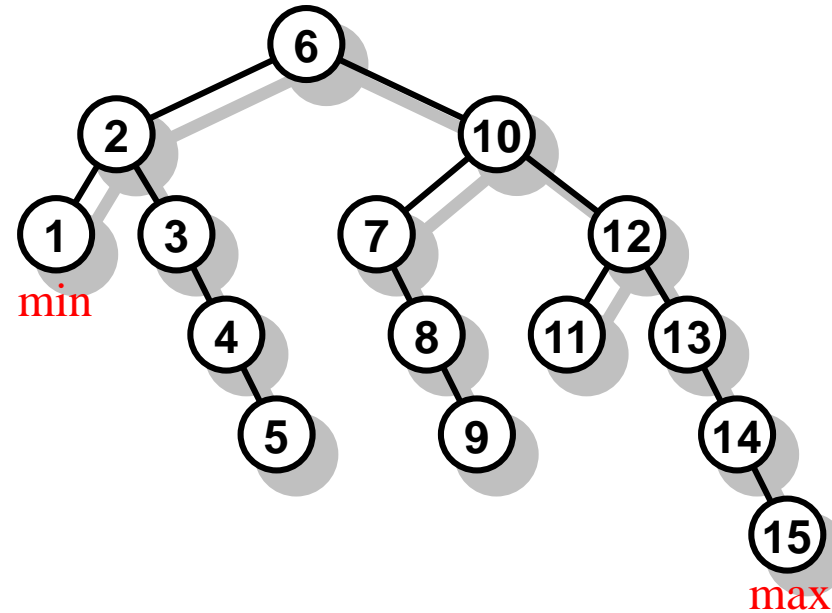
```
1  if  $x == \text{NIL}$  or  $k == x.\text{key}$ 
2      return  $x$ 
3  if  $k < x.\text{key}$ 
4      return TREE-SEARCH( $x.\text{left}, k$ )
5  else return TREE-SEARCH( $x.\text{right}, k$ )
```

ITERATIVE-TREE-SEARCH( $x, k$ )

```
1  while  $x \neq \text{NIL}$  and  $k \neq x.\text{key}$ 
2      if  $k < x.\text{key}$ 
3           $x = x.\text{left}$ 
4      else  $x = x.\text{right}$ 
5  return  $x$ 
```

TREE-SUCCESSOR( $x$ )

```
1  if  $x.\text{right} \neq \text{NIL}$ 
2      return TREE-MINIMUM( $x.\text{right}$ )
3   $y = x.p$ 
4  while  $y \neq \text{NIL}$  and  $x == y.\text{right}$ 
5       $x = y$ 
6       $y = y.p$ 
7  return  $y$ 
```



INORDER-TREE-WALK( $x$ )

```
1  if  $x \neq \text{NIL}$ 
2      INORDER-TREE-WALK( $x.\text{left}$ )
3      print  $x.\text{key}$ 
4      INORDER-TREE-WALK( $x.\text{right}$ )
```

TREE-MINIMUM( $x$ )

```
1  while  $x.\text{left} \neq \text{NIL}$ 
2       $x = x.\text{left}$ 
3  return  $x$ 
```

TREE-MAXIMUM( $x$ )

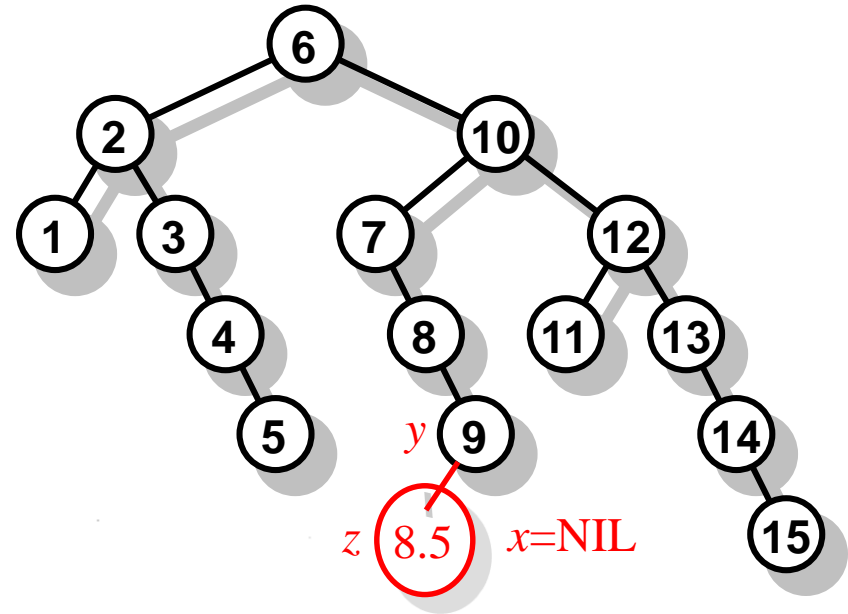
```
1  while  $x.\text{right} \neq \text{NIL}$ 
2       $x = x.\text{right}$ 
3  return  $x$ 
```

# Indsættelse i Søgetræ

TREE-INSERT( $T, z$ )

Iterativ søgning

```
1   $y = \text{NIL}$ 
2   $x = T.\text{root}$ 
3  while  $x \neq \text{NIL}$ 
4       $y = x$ 
5      if  $z.\text{key} < x.\text{key}$ 
6           $x = x.\text{left}$ 
7      else  $x = x.\text{right}$ 
8   $z.p = y$ 
9  if  $y == \text{NIL}$ 
10      $T.\text{root} = z$  // tree  $T$  was empty
11  elseif  $z.\text{key} < y.\text{key}$ 
12      $y.\text{left} = z$ 
13  else  $y.\text{right} = z$ 
```

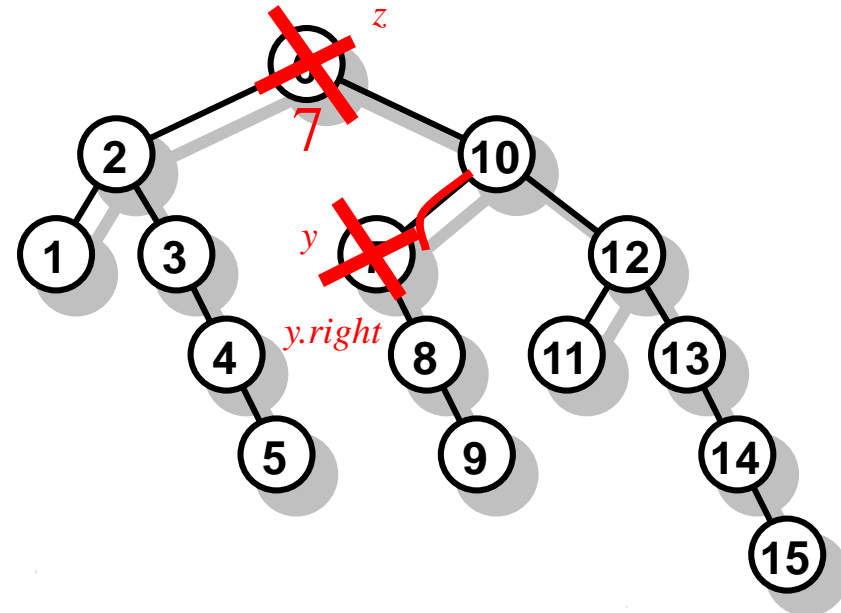




# Slettelse fra Søgetræ [CLRS, Ed. 3]

TREE-DELETE( $T, z$ )

```
1  if  $z.left == \text{NIL}$ 
2      TRANSPLANT( $T, z, z.right$ )
3  elseif  $z.right == \text{NIL}$ 
4      TRANSPLANT( $T, z, z.left$ )
5  else  $y = \text{TREE-MINIMUM}(z.right)$ 
6      if  $y.p \neq z$ 
7          TRANSPLANT( $T, y, y.right$ )
8           $y.right = z.right$ 
9           $y.right.p = y$ 
10     TRANSPLANT( $T, z, y$ )
11      $y.left = z.left$ 
12      $y.left.p = y$ 
```



TRANSPLANT( $T, u, v$ )

```
1  if  $u.p == \text{NIL}$ 
2       $T.root = v$ 
3  elseif  $u == u.p.left$ 
4       $u.p.left = v$ 
5  else  $u.p.right = v$ 
6  if  $v \neq \text{NIL}$ 
7       $v.p = u.p$ 
```

# Søgetræer

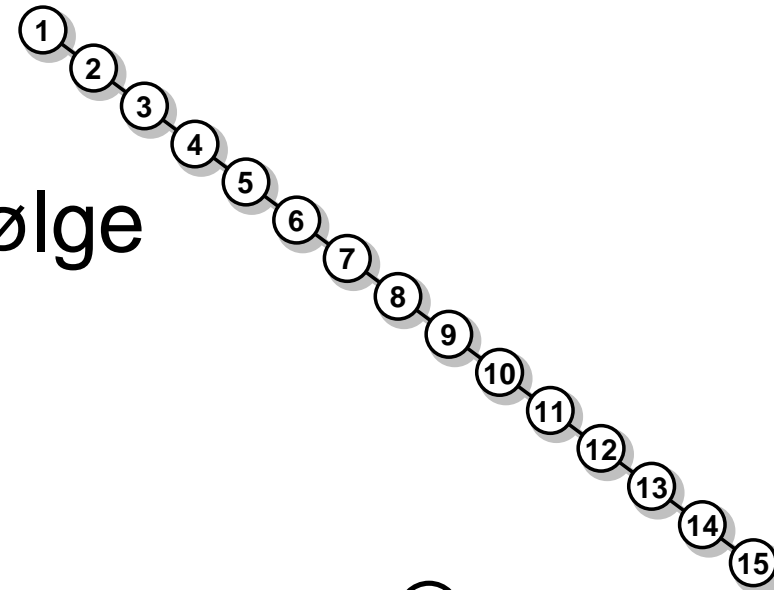
<b>Inorder-Tree-Walk</b>	$O(n)$
<b>Tree-Search</b> <b>Iterative-Tree-Search</b>	$O(h)$
<b>Tree-Minimum</b> <b>Tree-Maximum</b>	$O(h)$
<b>Tree-Insert</b>	$O(h)$
<b>Tree-Delete</b>	$O(h)$

$h$  = højden af træet,  $n$  = antal elementer

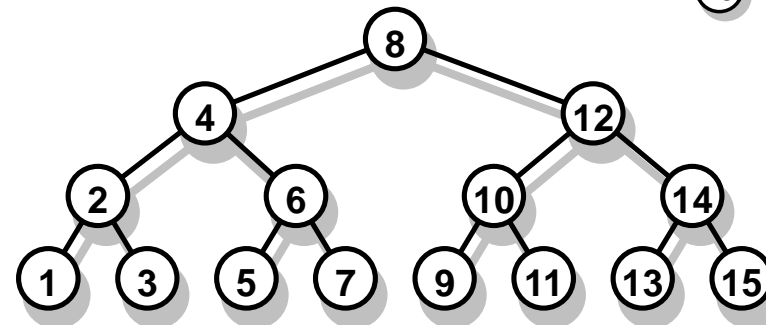
**Højden af et Søgetræ ?**

# Største og Mindste Højde

Indsæt i stigende rækkefølge  
- Højde  $n$



Perfekt balanceret  
(mindst mulig højde)  
- Højde  $1 + \lfloor \log n \rfloor$



# Tilfældige Indsættelser i et Søgetræ

## Algoritme:

Indsæt  $n$  elementer i tilfældige rækkefølge

- Ligesom ved QuickSort argumenteres at den **forventede dybde** af et element er  $O(\log n)$
- Den forventede **højde af træet** er  $O(\log n)$ , dvs. *alle knuder* har forventet dybde  $O(\log n)$   
[CLRS, Kap. 12.4]



# Balancerede Søgetræer

Ved **balancerede søgetræer** omstruktureres træet løbende så søgetiderne forbliver  $O(\log n)$

- AVL-træer [CLRS pr. 13-3]
- BB[ $\alpha$ ]-træer
- Splay træer
- Lokalt balancerede træer
- Rød-sorter træer [CLRS kap. 13]
- Randomized trees [CLRS pr. 13-4]
- (2,3)-træer
- (2,4)-træer [CLRS pr. 18-2]
- B-træer [CLRS kap. 18]
- Vægtbalancerede B-træer
- ...