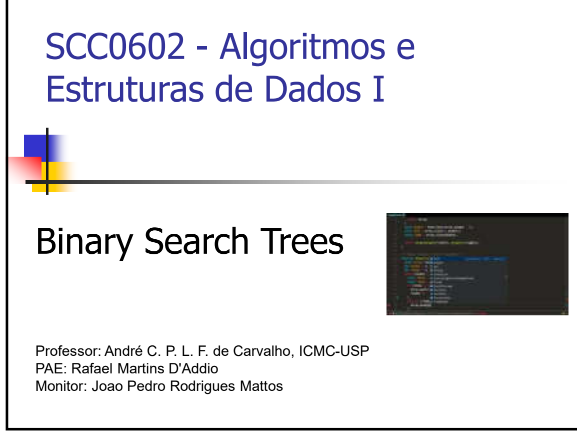
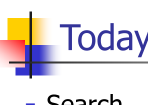




Today

- Search
- Linear search
- Binary Search Trees
  - Tree traversals (using divide-and-conquer)
  - Searching
  - Insertion
  - Deletion

© André de Carvalho - ICMC/USP

Introduction

- Search is frequently in several applications
  - Games
    - Best players in checker and chess are search algorithms
  - Minimum path
    - Traveller salesman problem are solved by search algorithms
  - Search engines
    - Search algorithms can find the most relevant sites
    - Dictionaries

© André de Carvalho - ICMC/USP

Chess

- In 1997, the current world champion Gary Kasparov played 6 games against Deep Blue, a program written by IBM researchers
  - Deep Blue won 3, lost 2, tied 1
    - Searched 126.000.000 nodes per sec
    - Generated 30 billion positions per reaching depth 14 routinely

© André de Carvalho - ICMC/USP

Dictionaries

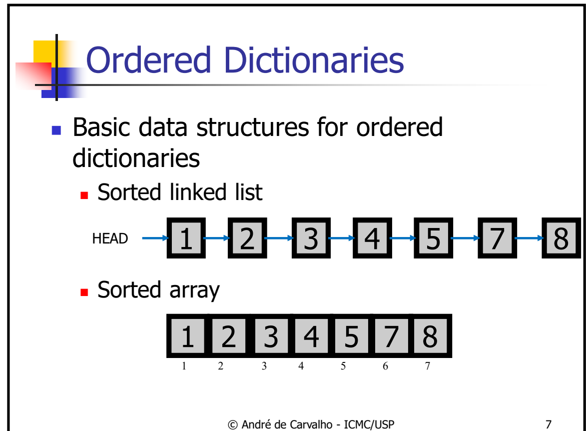
- *Dictionary* Abstract data type (ADT)
- Dynamic set with methods:
  - **Search(S, k)** – *a query method that returns a pointer x to an element, where x.key = k*
  - **Insert(S, x)** – *a modifier method that adds the element pointed to by x to S*
  - **Delete(S, x)** – *a modifier method that removes the element pointed to by x from S*
- An element has a *key* part and a *satellite data* part

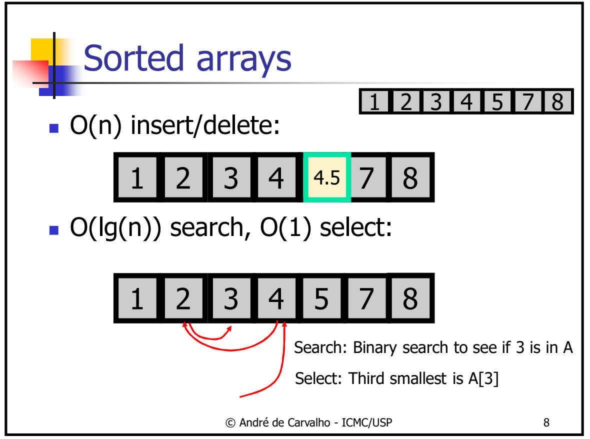
© André de Carvalho - ICMC/USP

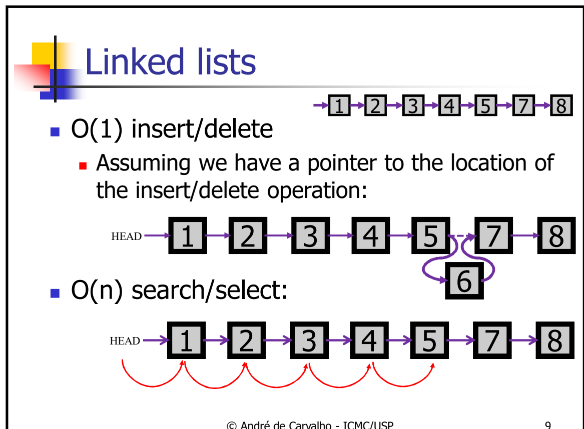
Ordered Dictionaries

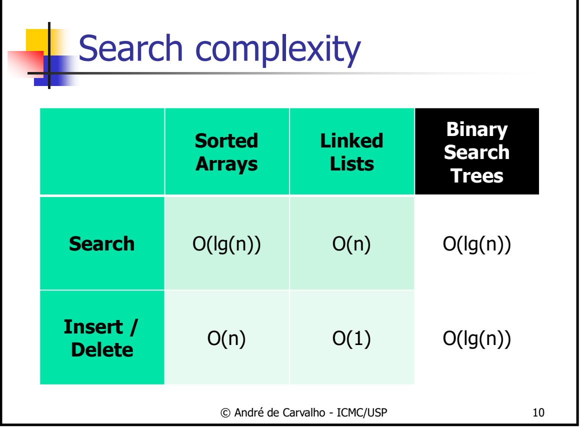
- Besides the previous functions, it should also support the priority-queue-type operations
  - **Min(S)**
  - **Max(S)**
- It would be useful to support
  - **Predecessor(S, k)**
  - **Successor(S, k)**
- These operations require the keys to be comparable

© André de Carvalho - ICMC/USP

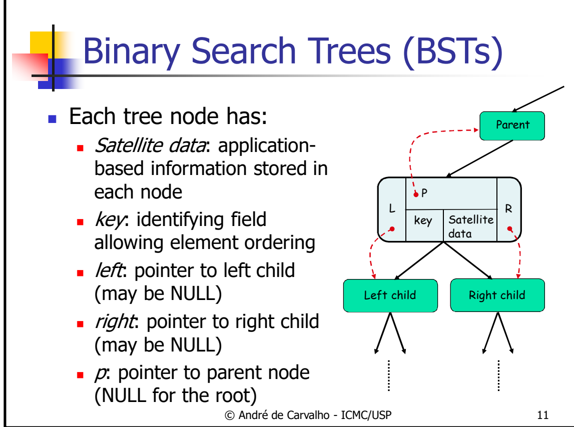


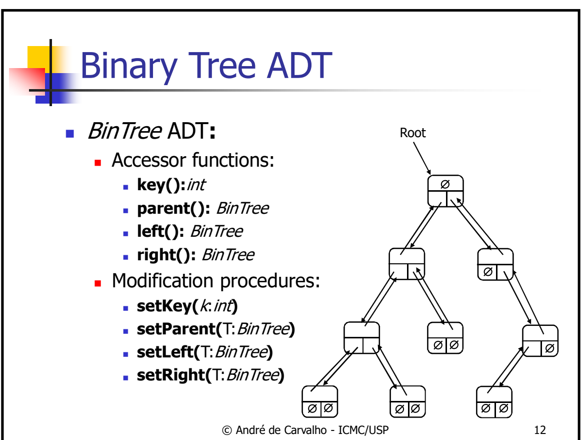


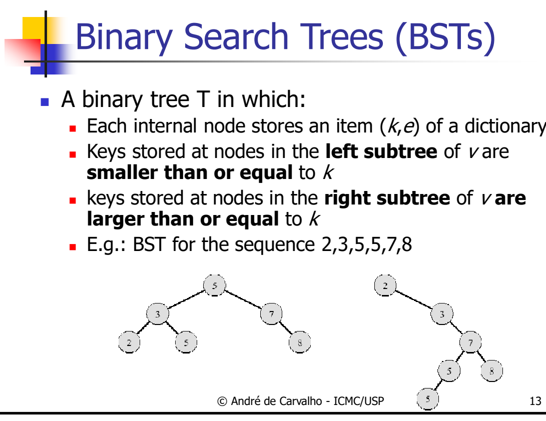


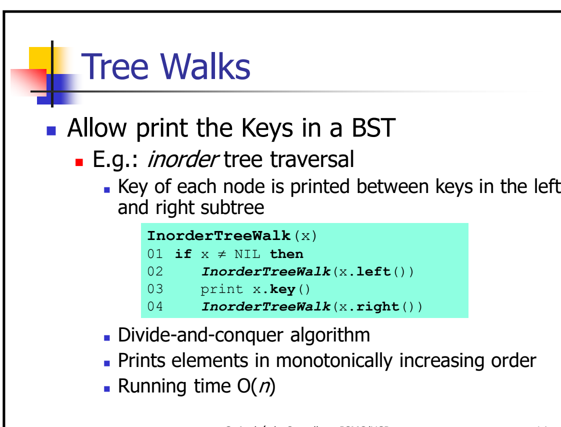


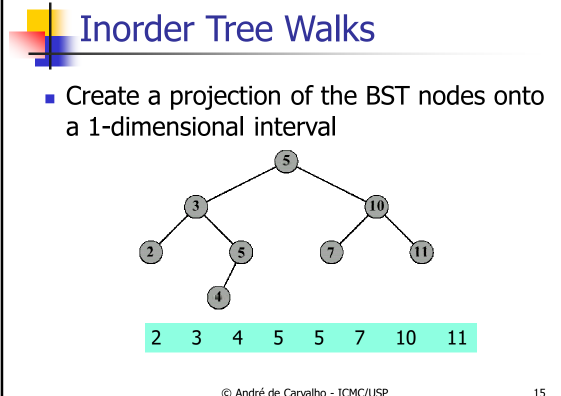
	Sorted Arrays	Linked Lists	Binary Search Trees
Search	O(lg(n))	O(n)	O(lg(n))
Insert / Delete	O(n)	O(1)	O(lg(n))

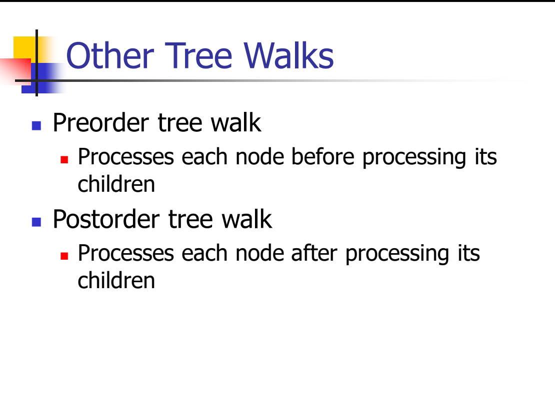


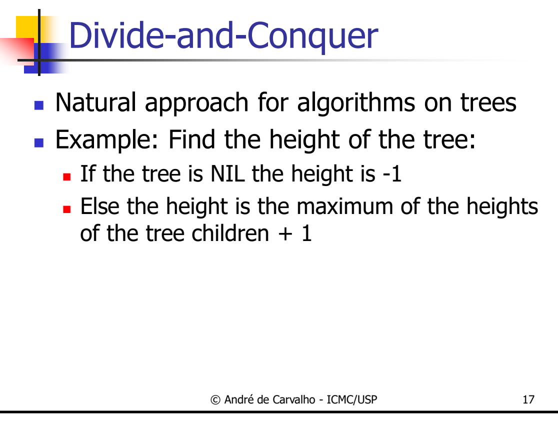


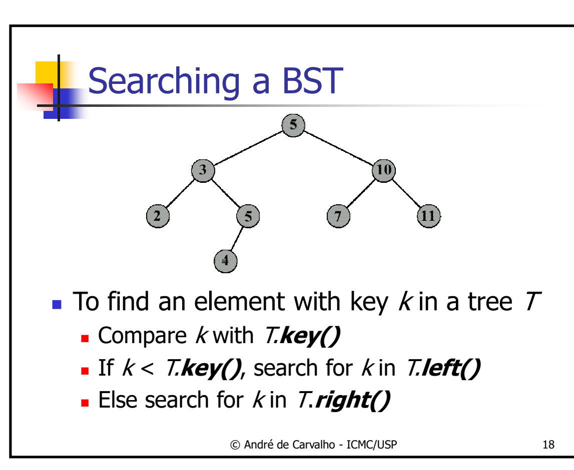

Binary Search Trees (BSTs)
A binary tree T in which:
Each internal node stores an item (k,e) of a dictionary
Keys stored at nodes in the left subtree of v are smaller than or equal to k
keys stored at nodes in the right subtree of v are larger than or equal to k
E.g.: BST for the sequence 2,3,5,5,7,8
© André de Carvalho - ICMC/USP
13

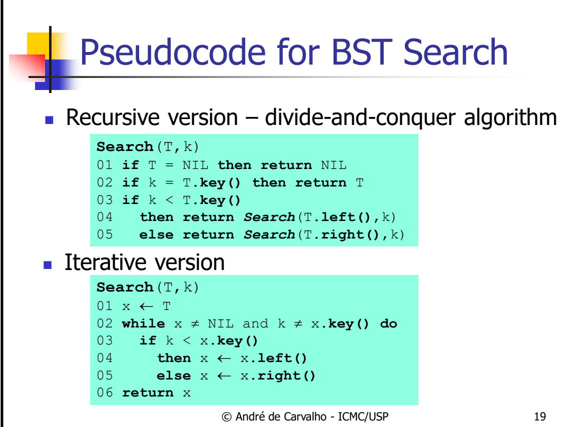

Tree Walks
Allow print the Keys in a BST
E.g.: inorder tree traversal
Key of each node is printed between keys in the left and right subtree
InorderTreeWalk(x)
01 if x ≠ NIL then
02 InorderTreeWalk(x.left())
03 print x.key()
04 InorderTreeWalk(x.right())
Divide-and-conquer algorithm
Prints elements in monotonically increasing order
Running time O(n)
© André de Carvalho - ICMC/USP
14

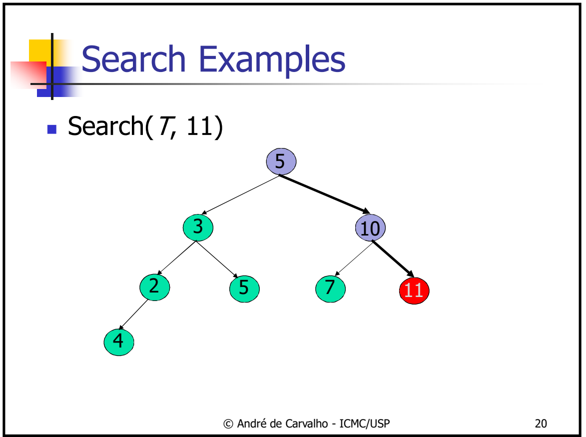

Inorder Tree Walks
Create a projection of the BST nodes onto a 1-dimensional interval
5
3
10
2
5
7
11
4
2 3 4 5 5 7 10 11
© André de Carvalho - ICMC/USP
15

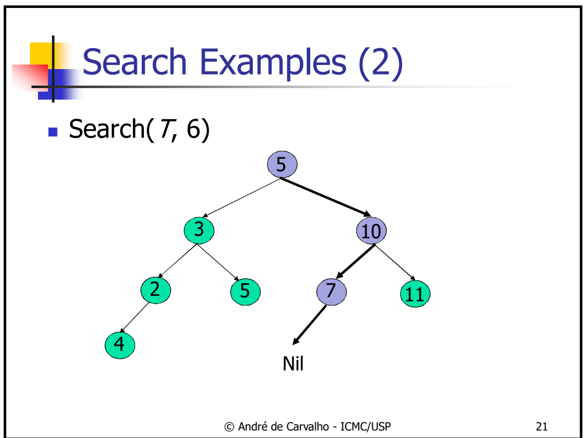

Other Tree Walks
Preorder tree walk
Processes each node before processing its children
Postorder tree walk
Processes each node after processing its children
© André de Carvalho - ICMC/USP
16

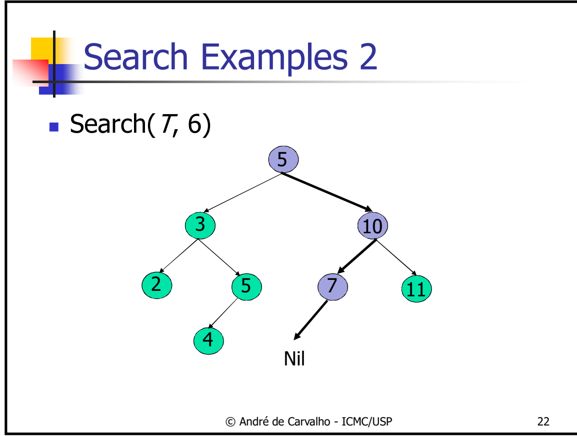

Divide-and-Conquer
Natural approach for algorithms on trees
Example: Find the height of the tree:
If the tree is NIL the height is -1
Else the height is the maximum of the heights of the tree children + 1
© André de Carvalho - ICMC/USP
17

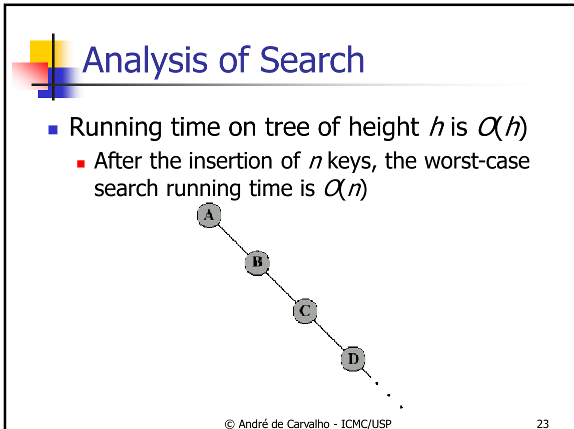

Searching a BST
5
3
10
2
5
7
11
4
To find an element with key k in a tree T
Compare k with T.key()
If k < T.key(), search for k in T.left()
Else search for k in T.right()
© André de Carvalho - ICMC/USP
18

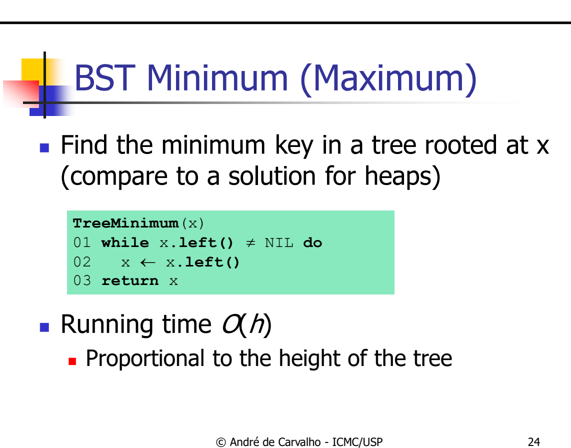

Pseudocode for BST Search
Recursive version – divide-and-conquer algorithm
Search(T,k)
01 if T = NIL then return NIL
02 if k = T.key() then return T
03 if k < T.key()
04 then return Search(T.left(),k)
05 else return Search(T.right(),k)
Iterative version
Search(T,k)
01 x ← T
02 while x ≠ NIL and k ≠ x.key() do
03 if k < x.key()
04 then x ← x.left()
05 else x ← x.right()
06 return x
© André de Carvalho - ICMC/USP
19


Search Examples
Search(T, 11)
5
3
10
2
5
7
11
4
© André de Carvalho - ICMC/USP
20


Search Examples (2)
Search(T, 6)
5
3
10
2
5
7
11
4
Nil
© André de Carvalho - ICMC/USP
21

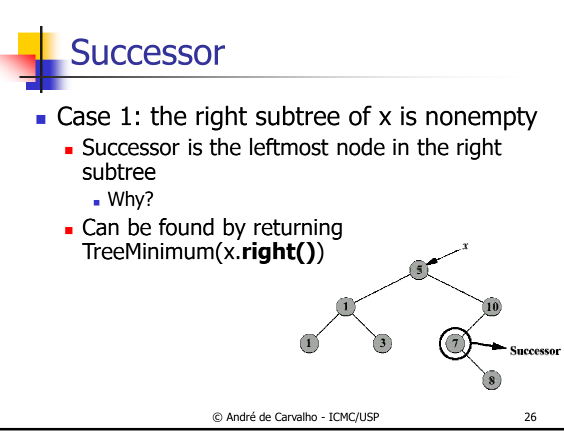

Search Examples 2
Search(T, 6)
5
3
10
2
5
7
11
4
Nil
© André de Carvalho - ICMC/USP
22

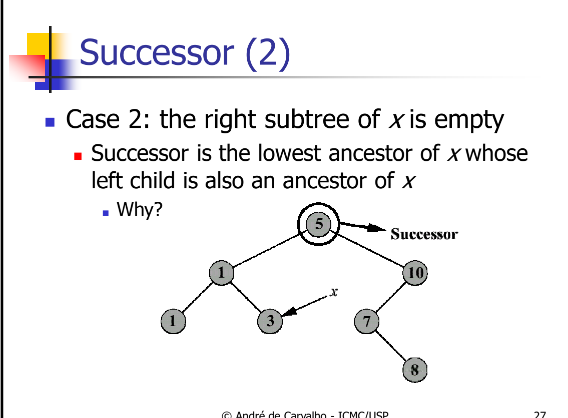

Analysis of Search
Running time on tree of height h is O(h)
After the insertion of n keys, the worst-case search running time is O(n)
A
B
C
D
© André de Carvalho - ICMC/USP
23

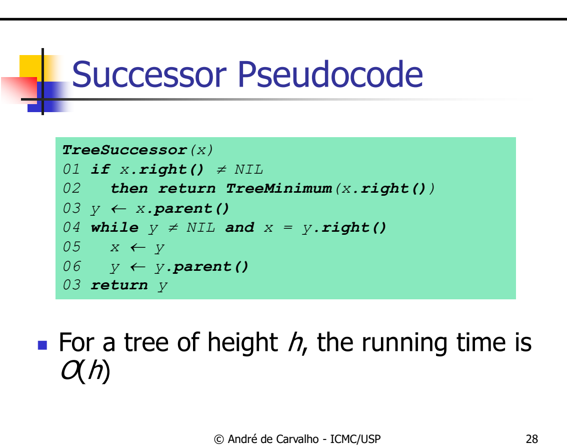

BST Minimum (Maximum)
Find the minimum key in a tree rooted at x (compare to a solution for heaps)
TreeMinimum(x)
01 while x.left() ≠ NIL do
02 x ← x.left()
03 return x
Running time O(h)
Proportional to the height of the tree
© André de Carvalho - ICMC/USP
24

Successor

- Given *x*, find the node with the smallest key that is larger than *x*.**key()**
- There are two possible cases, depending on the right subtree of *x*
  - Case 1: the right subtree of x is nonempty
  - Case 2: the right subtree of *x* is empty





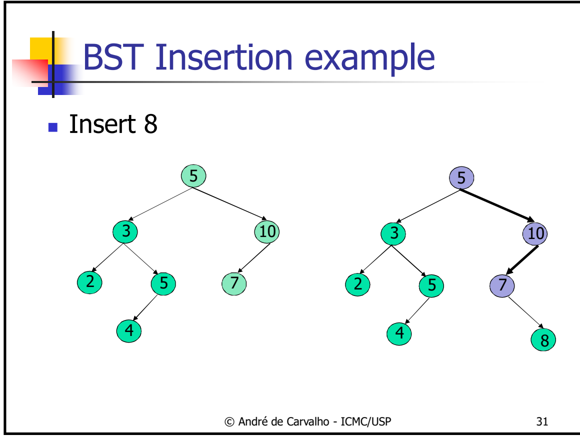


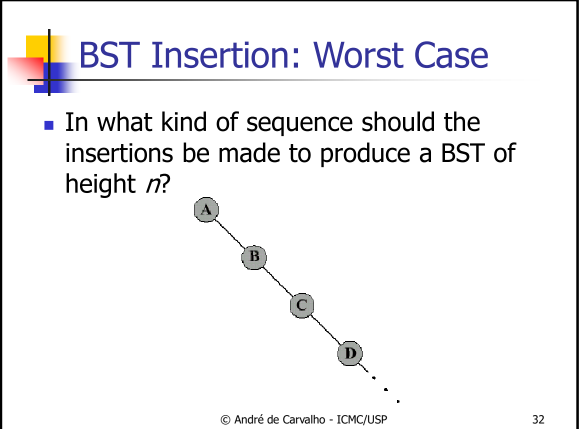
BST Insertion

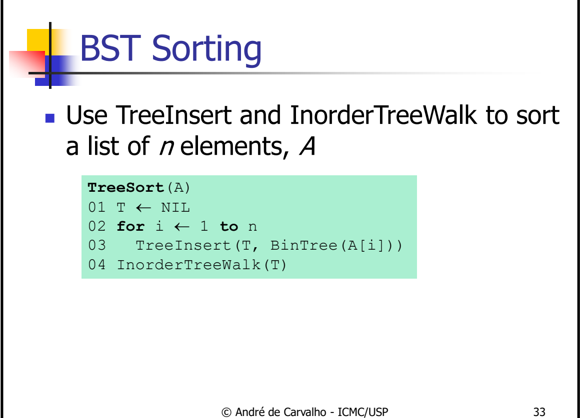
- The basic idea is similar to searching
  - Take an element (tree) *z* (whose left and right children are NIL) and insert it into *T*
  - Find place in *T* where *z* belongs, as if searching for *z.key()*, and add *z* there
- The running on a tree of height *h* is $O(h)$

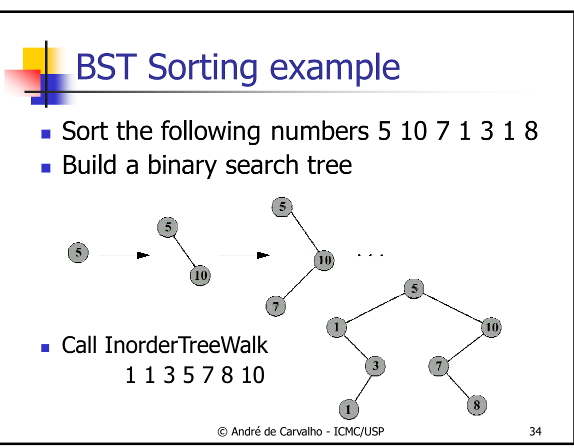
BST Insertion Pseudo Code

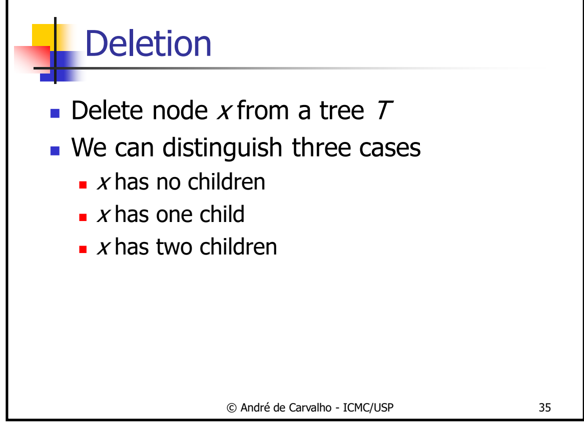
```
TreeInsert(T,z)
01 y ← NIL
02 x ← T
03 while x ≠ NIL
04   y ← x
05   if z.key() < x.key()
06     then x ← x.left()
07     else x ← x.right()
08 z.setParent(y)
09 if y ≠ NIL
10   then if z.key() < z.key()
11          then y.setLeft(z)
12          else y.setRight(z)
13   else T ← z
```

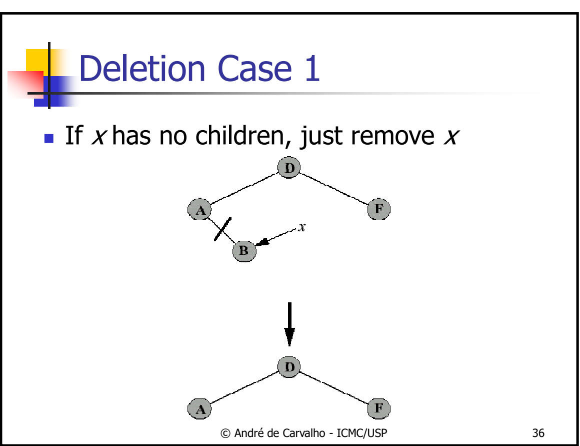

BST Insertion example
Insert 8
5
3
10
2
5
7
4
5
3
10
2
5
7
4
8
© André de Carvalho - ICMC/USP
31

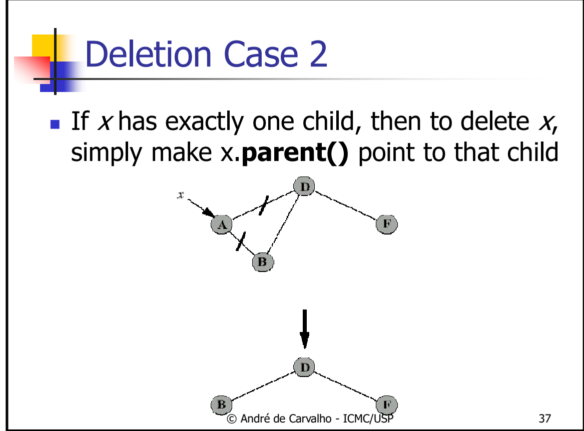

BST Insertion: Worst Case
In what kind of sequence should the insertions be made to produce a BST of height n?
A
B
C
D
© André de Carvalho - ICMC/USP
32

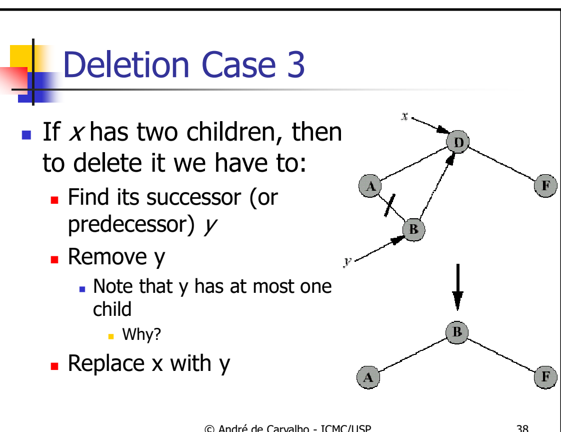

BST Sorting
Use TreeInsert and InorderTreeWalk to sort a list of n elements, A
TreeSort(A)
01 T ← NIL
02 for i ← 1 to n
03   TreeInsert(T, BinTree(A[i]))
04 InorderTreeWalk(T)
© André de Carvalho - ICMC/USP
33

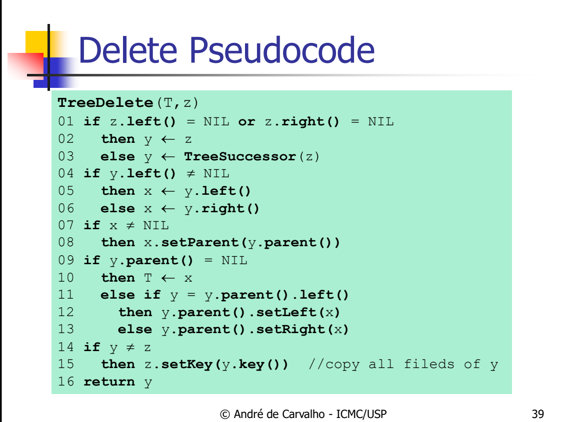

BST Sorting example
Sort the following numbers 5 10 7 1 3 1 8
Build a binary search tree
Call InorderTreeWalk
1 1 3 5 7 8 10
© André de Carvalho - ICMC/USP
34

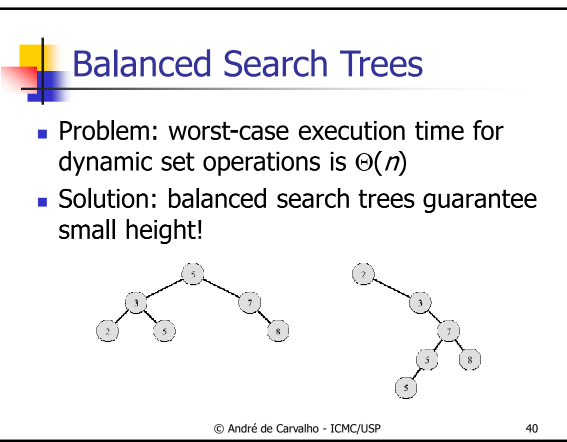

Deletion
Delete node x from a tree T
We can distinguish three cases
x has no children
x has one child
x has two children
© André de Carvalho - ICMC/USP
35


Deletion Case 1
If x has no children, just remove x
D
A
F
B
x
D
A
F
© André de Carvalho - ICMC/USP
36

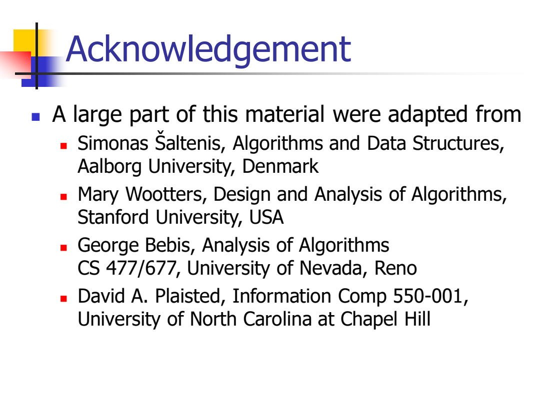

Deletion Case 2
If x has exactly one child, then to delete x, simply make x.parent() point to that child
x
A
B
D
F
B
D
F
© André de Carvalho - ICMC/USP
37

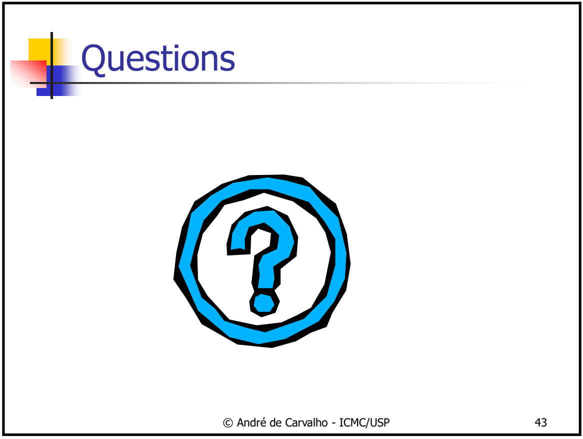

Deletion Case 3
If x has two children, then to delete it we have to:
Find its successor (or predecessor) y
Remove y
Note that y has at most one child
Why?
Replace x with y
x
D
A
F
B
y
B
A
F
© André de Carvalho - ICMC/USP
38


Delete Pseudocode
TreeDelete(T,z)
01 if z.left() = NIL or z.right() = NIL
02 then y ← z
03 else y ← TreeSuccessor(z)
04 if y.left() ≠ NIL
05 then x ← y.left()
06 else x ← y.right()
07 if x ≠ NIL
08 then x.setParent(y.parent())
09 if y.parent() = NIL
10 then T ← x
11 else if y = y.parent().left()
12 then y.parent().setLeft(x)
13 else y.parent().setRight(x)
14 if y ≠ z
15 then z.setKey(y.key()) //copy all fileds of y
16 return y
© André de Carvalho - ICMC/USP
39


Balanced Search Trees
Problem: worst-case execution time for dynamic set operations is Θ(n)
Solution: balanced search trees guarantee small height!
© André de Carvalho - ICMC/USP
40

Next Lecture
Hashing
© André de Carvalho - ICMC/USP
41


Acknowledgement
A large part of this material were adapted from
Simonas Šaltenis, Algorithms and Data Structures, Aalborg University, Denmark
Mary Wootters, Design and Analysis of Algorithms, Stanford University, USA
George Bebis, Analysis of Algorithms CS 477/677, University of Nevada, Reno
David A. Plaisted, Information Comp 550-001, University of North Carolina at Chapel Hill
© André de Carvalho - ICMC/USP
42


Questions
© André de Carvalho - ICMC/USP
43