CENG 218
Design and Analysis of Algorithms

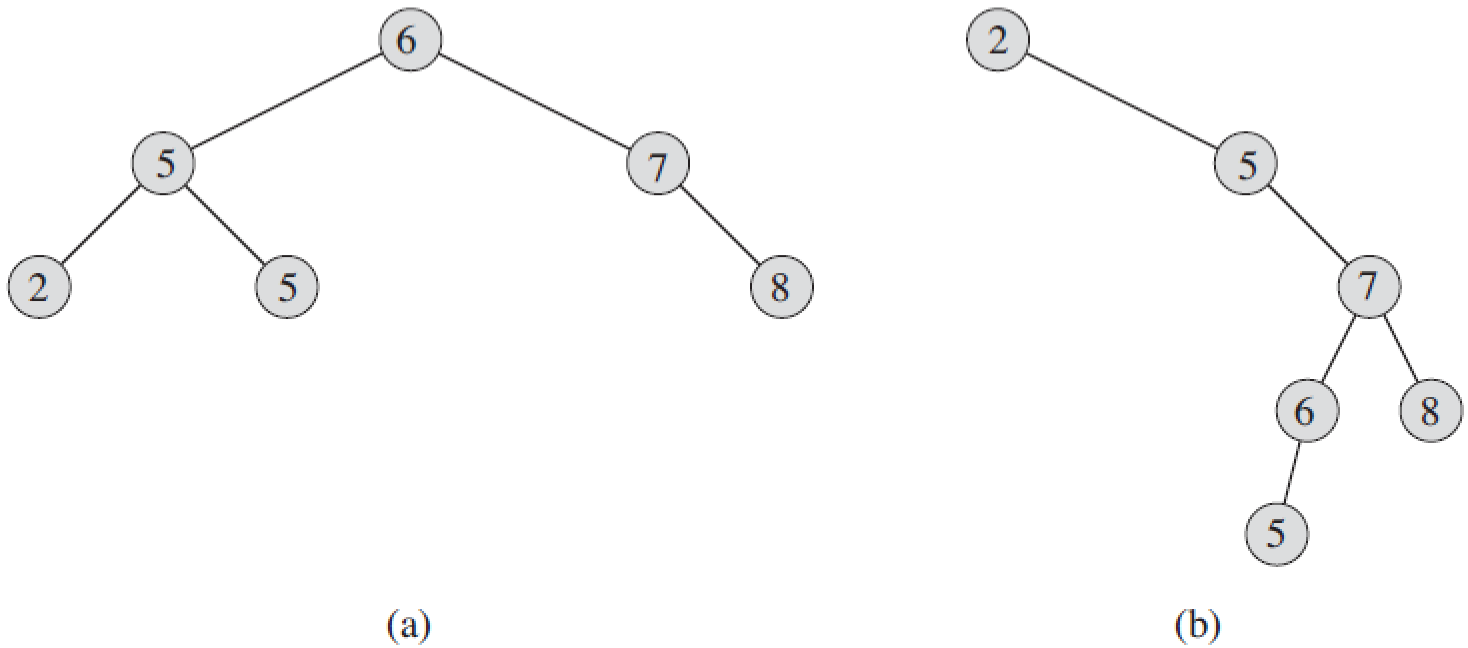
Izmir Institute of Technology

Lecture 9: Binary search trees

Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

What is a binary search tree (BST)?

For any node *x*, the keys in left subtree of *x* are at most *x*.*key*, and the keys in right subtree of *x* are at least *x*.*key*.



(a) A binary search tree on 6 nodes with height 2.
(b) A less efficient binary search tree with height 4 that contains the same keys.

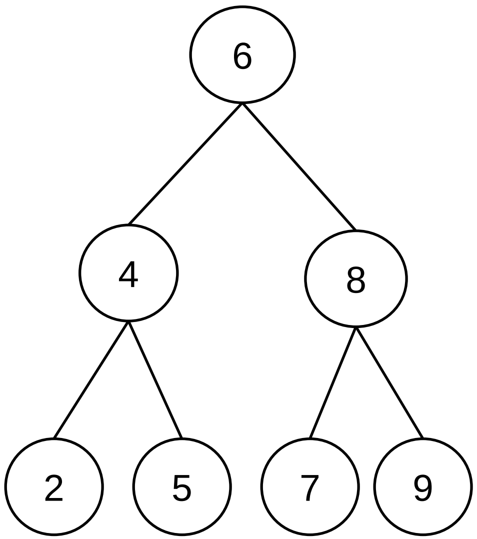
Inorder tree walk

It is a simple recursive algorithm to print all the keys in a BST in sorted order.
It prints the key of the root of a subtree between the values in its left subtree and its right subtree.

INORDER-TREE-WALK (x)
  if $x \neq$ NIL
    INORDER-TREE-WALK ($x.left$)
    print $x.key$
    INORDER-TREE-WALK ($x.right$)

Similarly, a ***preorder tree walk*** prints the root before the values in either subtree.

Tree walk (traversal) examples



In-order tree traversal: 2456789

Pre-order tree traversal: 6425879

Post-order tree traversal: 2547986

Binary-search-tree sort

Q. How can we sort using a BST?
A. Once we build a BST, we perform Inorder-tree-walk with running time = $\Theta(n)$.

Q. How long does it take to build the BST?
A. BST is built by n insertion operations:

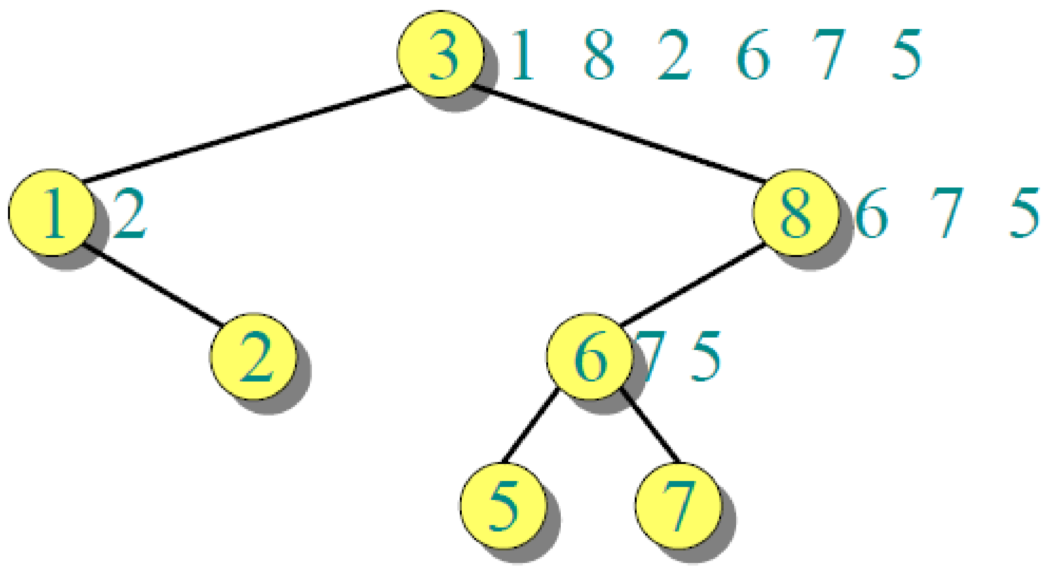
$T \leftarrow \emptyset$ {Create an empty BST}
for i = 1 to n
  do TREE-INSERT(T, $A[i]$) {A is the unsorted list}

Q. How long does it take one insertion operation?
A. $\Omega(\lg n)$ ..will see in a minute..

Binary-search-tree sort example

A = [3 1 8 2 6 7 5]

BST sort performs the same comparisons as Quicksort, but in a different order!



Randomized BST sort

1) Randomly permute A
2) BST sort (A)

The expected time to build the tree is asymptotically the same as the running time of Quicksort which is $\Theta(n \lg n)$.

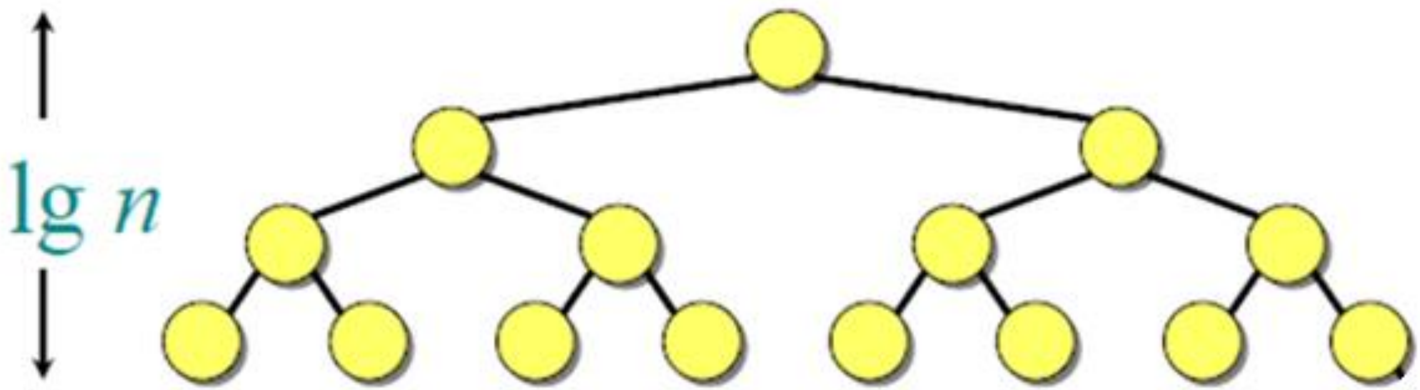
BST

Although we started with BST-sort, main use of BST is not sorting.

It is an essential symbol-table implementation.
(Remember symbol-table problem, keep a key-value per record, duplicate keys are not allowed, like student lists, phone books)

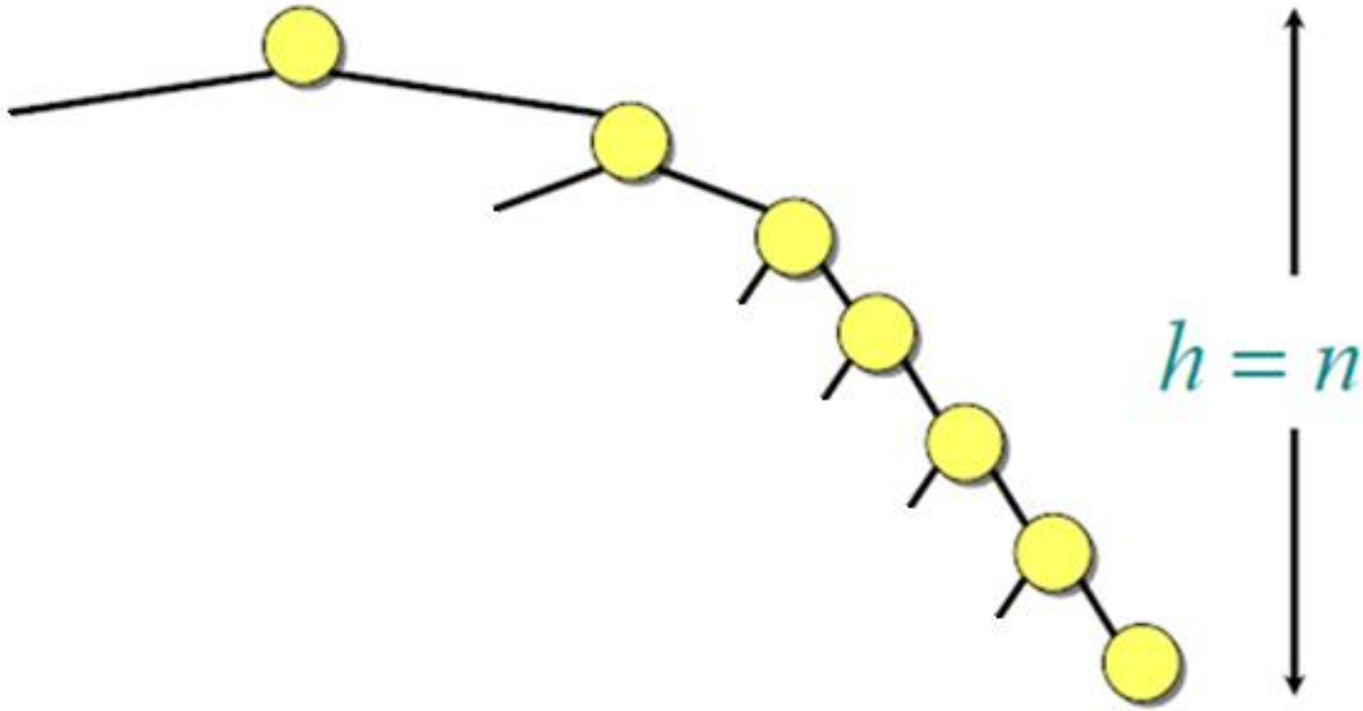
Main operations are search, insert and delete.
How long does it take to perform these operations?

BST best case



Search and insert time is $O(\lg n)$.

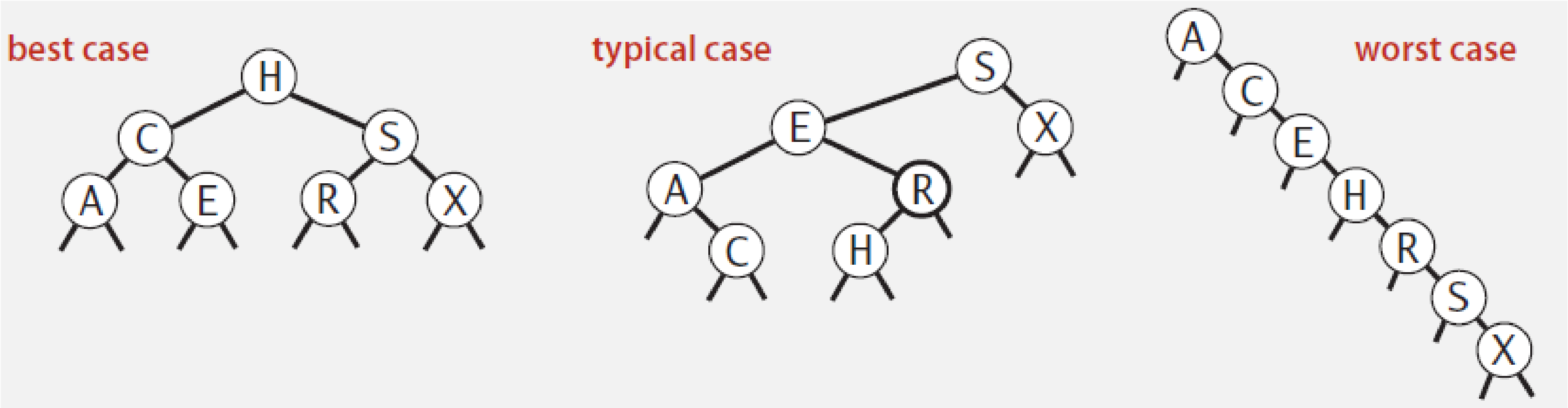
BST worst case



Search and insert time is $O(n)$.

BST best, typical, worst case

Another example (with key values).



How long does it take for the typical case?

Figure source: Algorithms, 4th edition by Sedgewick and Wayne

BST insertion:
a random order simulation

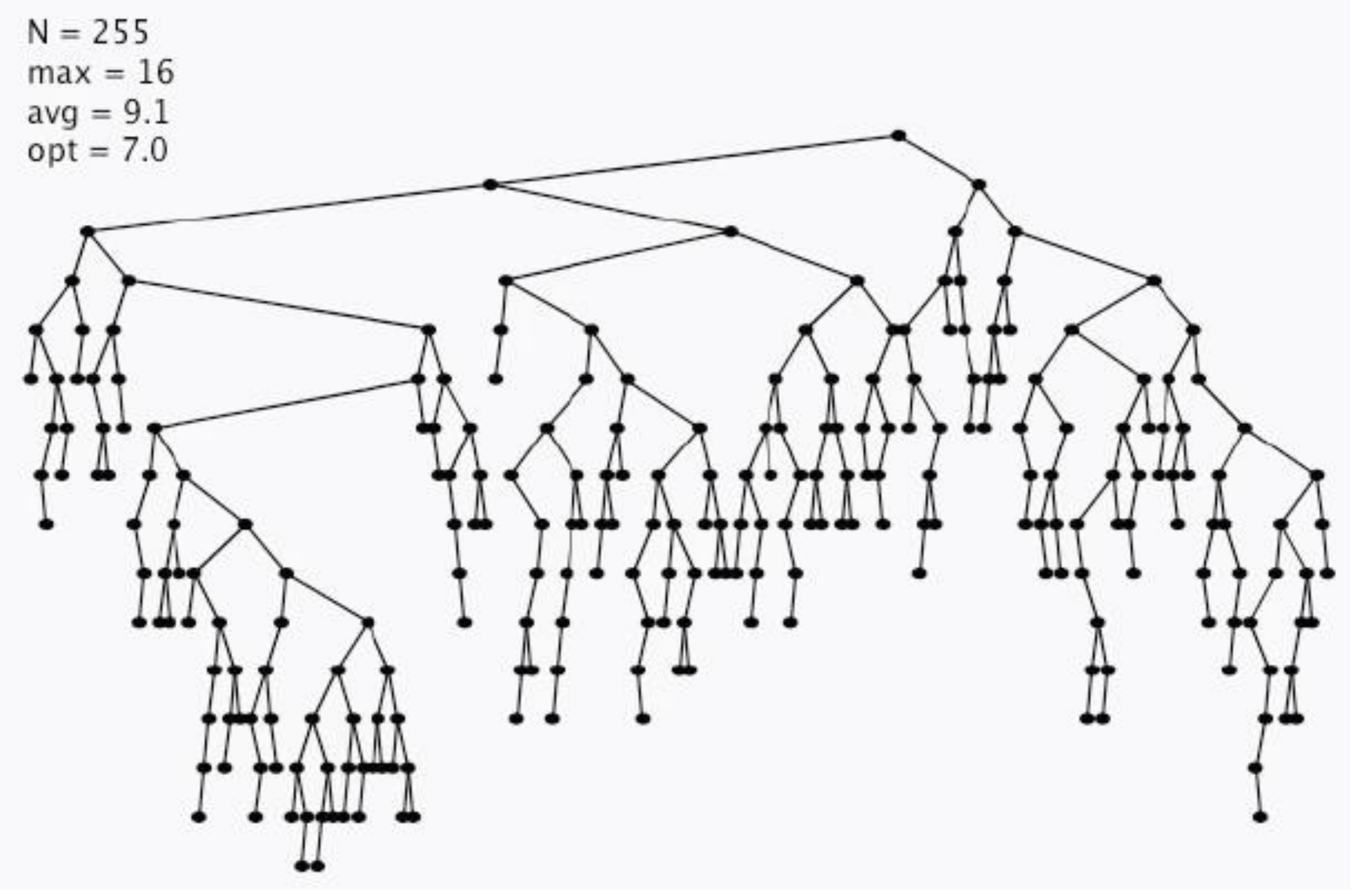


Figure source: Algorithms, 4th edition by Sedgewick and Wayne

Comparison of elementary symbol-table implementations

implementation	worst-case cost (after N inserts)			average case (after N random inserts)		
	search	insert	delete	search hit	insert	delete
sequential search (unordered list)	N	N	N	N/2	N	N/2
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2
BST	N	N	N	1.39 lg N	1.39 lg N	?

This is $\sqrt{N}$ even with an efficient method

Figure source: Algorithms, 4th edition by Sedgewick and Wayne

BST compared with hash tables

Remember that hash table complexity is $\Theta(1)$. Why we need BST as an average case $\Theta(\log n)$ search/insert implementation of symbol tables?

- It is hard to estimate the optimum number of slots in hash tables. They may reserve more memory than they need to. BSTs are memory-efficient.
- Depending on the load factor α, complexity of $\Theta(1)$ may be hard to achieve with hash tables.
- When needed, binary tree can be traversed to list the elements in order (BST sort).
- Range search can be done efficiently with BST.

The End

BSTs are in Section 12 of the textbook.

Next, we will continue with balanced search trees which guarantee worst-case $\Theta(\log n)$.