

# CSC 212: Data Structures and Abstractions

## Binary search trees (part 2)

Prof. Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2025



## Problems from lab

▸ Problem A

▸ Problem B

▸ Problem C

2

## Operations

## Contains

▸ Algorithm

- ✓ start at root node
- ✓ if the search key matches the current node's key then found
- ✓ if search key is greater than current node's key
  - search on right child
- ✓ if search key is less than current node's
  - search on left child
- ✓ stop when current node is nullptr (not found)

▸ Time complexity

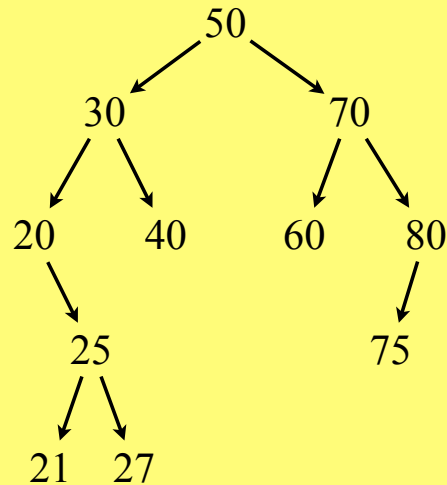
- ✓  $O(h)$ , where  $h$  is the height of the tree

4

## Practice

### Search the following keys:

✓ 25, 77, 18, 40, 75



5

## Insert

### Algorithm

- ✓ if tree is empty, create a new node as root, done
- ✓ if not, start node p at the root, repeat:
  - compare the key to insert with the key of p, if equal, done
  - if the new key is less than the key of p, set p to the left subtree
    - if p is empty, create new node here, done
    - else continue
  - if the new key is greater than the key of p, set p to the right subtree
    - if p is empty, create new node here, done
    - else continue

### Time complexity

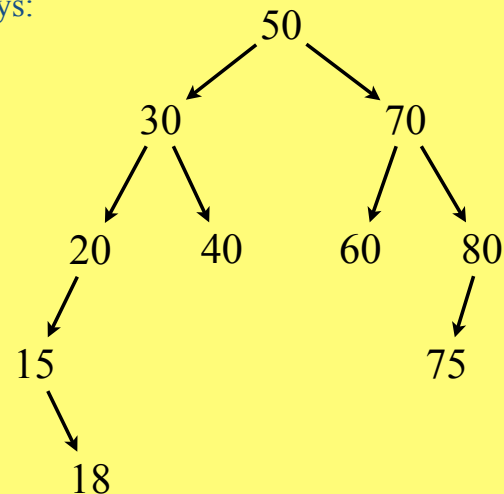
- ✓  $O(h)$ , where  $h$  is the height of the tree

6

## Practice

### Insert the following keys:

✓ 65, 27, 90, 11, 51



7

## Repeated keys

### We assume the tree contains unique keys

- ✓ no repeated keys are allowed

### Dealing with repeated keys

- ✓ if key is in the tree, do nothing, just return
- ✓ depending on the task being solved, may add a counter on each node, and it can be increased every time a repeated key is inserted
- ✓ if the tree is used as a map or dictionary, may want to update the value of a repeated key

8

## Remove

### Approach

- ✓ find node to be removed, then apply one of the cases below
- ✓ **case 1**: node is a leaf
  - trivial, delete node and set parent's pointer to nullptr
- ✓ **case 2**: node has 1 child
  - set parent's pointer to the only child and delete node
- ✓ **case 3**: node has 2 children
  - find successor (smallest node in the right subtree)
  - copy successor's data to node
  - delete successor

can also use  
predecessor

### Time complexity

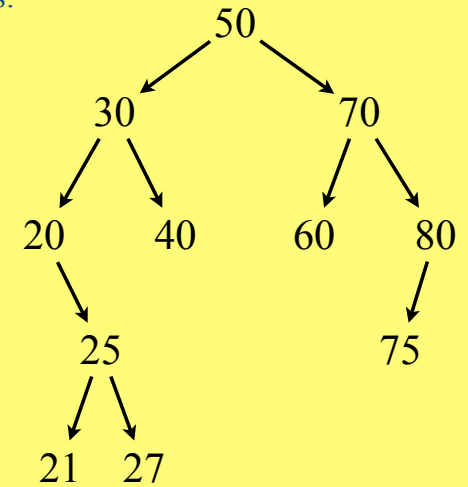
- ✓  $O(h)$ , where  $h$  is the height of the tree

9

## Practice

### Remove the following keys:

- ✓ 27, 40, 80, 20, 30, 50



10

## Analysis

## Practice

### Starting from an empty BST, insert the following keys in the order given

- ✓ 20 10 30 5 15 25 35

- ✓ 10 20 5 15 30 35 25

- ✓ 5 10 15 20 25 30 35

- ✓ How is the order of insertion related to the shape of the tree?
- ✓ How is the height of the tree related to the number of nodes?

12

## Practice

- Complete the following table with rates of growth
  - as a function of the number of nodes

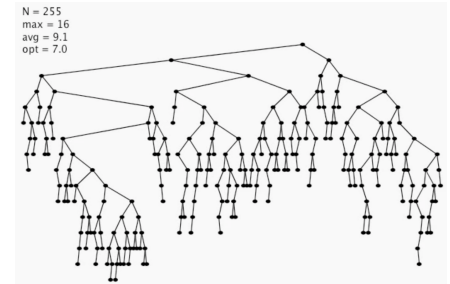
Operation	Best case	Average case	Worst case
Insert			
Remove			
Search			

13

## Average case

### Proposition

- if  $n$  distinct keys are randomly inserted into a BST, the expected number of compares is  $\sim c \log n$ 
  - can be formally justified through probabilistic analysis (not covered in this class)



### Implications

- even without explicit balancing mechanisms, randomly built BSTs provide reasonably efficient operations

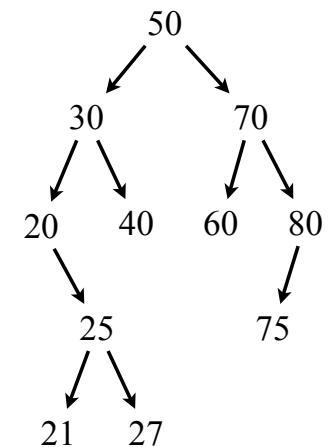
14

## Traversals

## Preorder traversal

- Depth-first traversal that visits the root node first, then recursively visits all subtrees

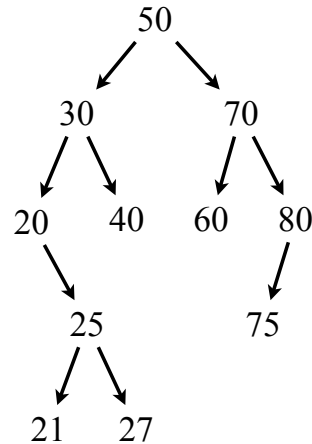
- visit the root node
- recursively visit the left subtree
- recursively visit the right subtree



16

## Postorder traversal

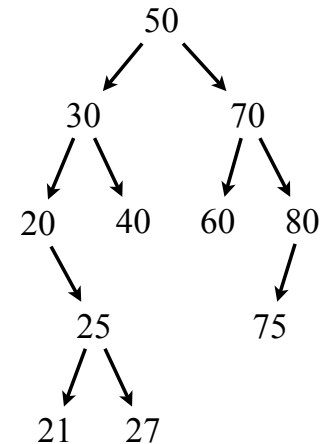
- Depth-first traversal that recursively visits all subtrees first, then visits the root node
  - recursively visit the left subtree
  - recursively visit the right subtree
  - visit the root node



17

## Inorder traversal

- Depth-first traversal that recursively visits the left subtree first, then visits the root node, and finally recursively visits the right subtree
  - recursively visit the left subtree
  - visit the root node
  - recursively visit the right subtree



18

## Practice

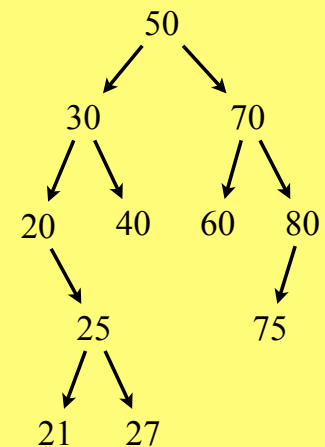
- Which traversal is best for printing all values in sorted order?
- Which traversal is best for deleting all nodes in a tree?
- What is the time complexity of each traversal?

19

## Practice

- Trace the following algorithm and explain what it does

```
algorithm mystery(root) {  
    queue q  
    q.enqueue(root)  
    while not q.isEmpty() {  
        node n = q.dequeue()  
        print(n.value)  
        if n.left  
            q.enqueue(n.left)  
        if n.right  
            q.enqueue(n.right)  
    }  
}
```



20

# Collections

Operation	Description	Sequential (unordered)	Sequential (ordered)	BST
search	search for a key	$O(n)$	$O(\log n)$	$O(h)$
insert	insert a key	$O(n)$	$O(n)$	$O(h)$
delete	delete a key	$O(n)$	$O(n)$	$O(h)$
min/max	find smallest/largest key	$O(n)$	$O(1)$	$O(h)$
floor/ceiling	find predecessor/successor	$O(n)$	$O(\log n)$	$O(h)$
rank	count number of keys less than key	$O(n)$	$O(\log n)$	$O(h)$