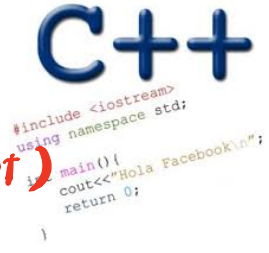
RUNNING TIME ANALYSIS OF BINARY SEARCH TREES

Problem Solving with Computers-II



PA01 → Monday (2nd Sept)

What is the Big O of sumArray2

A. $O(N^2)$
B. O(N)
C. O(N/2)
D. O(log N)
E. None of the array

```
/* N is the length of the array*/
int sumArray2(int arr[], int N)
{
    int result=0;
    for(int i=1; i < N; i=i*2)
        result+=arr[i];
    return result;
}
```

i = i + 2

$1 + 1 + 3 * (\log_2 N + 1)$

Running time of operations on sorted arrays: Discuss best case, worst case, average case

- Min : O(1)
- Max: O(1)
- Median: O(1)
- Successor: O(1)
- Predecessor: O(1)
- Search:
- Insert : O(N)
- Delete: O(N)

Naive linear search			Binary Search	
O(1)	O(N)		O(1)	O(log N)
Best case	Worst case	Avg. case	Best case	Worst case / Avg case

$C_1 + C_2 \log N$

Binary Search Trees

- WHAT are the operations supported?
- HOW do we implement them?
- WHAT are the (worst case) running times of each operation?

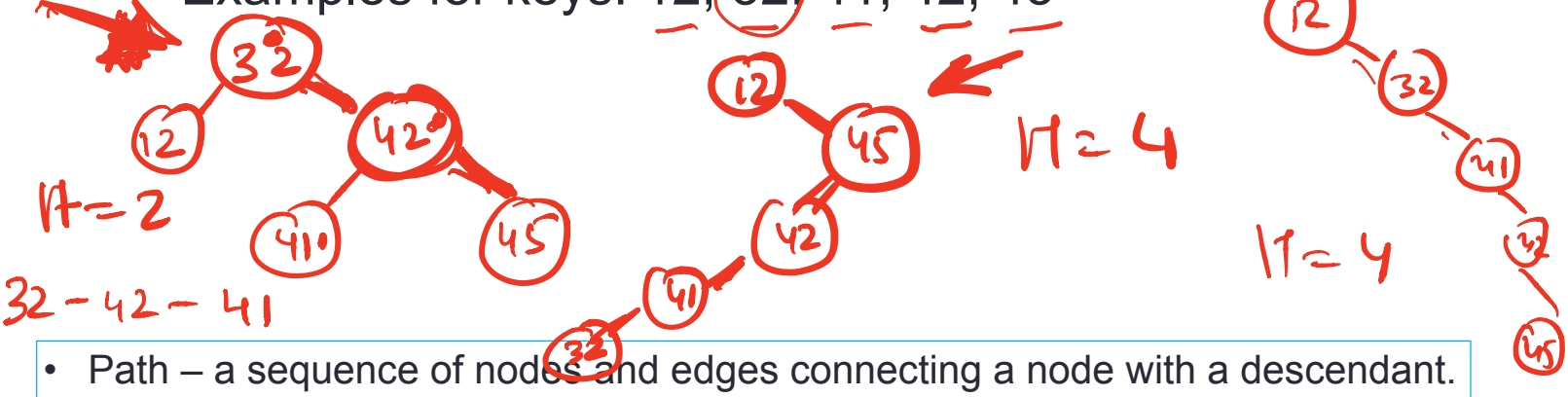
Visualize BST operations: https://visualgo.net/bn/bst

Height of the tree

Height

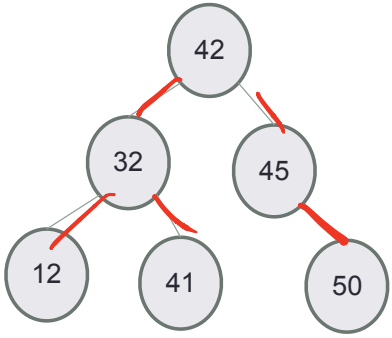
Many different BSTs are possible for the same set of keys

Examples for keys: 12, 32, 41, 42, 45



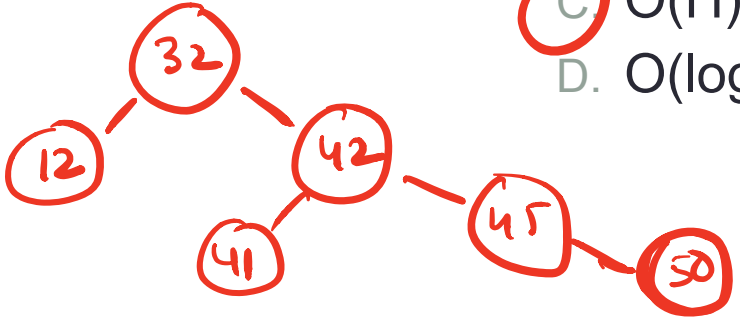
- Path – a sequence of nodes and edges connecting a node with a descendant.
- A path starts from a node and ends at another node or a leaf
- Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.

Worst case Big-O of search

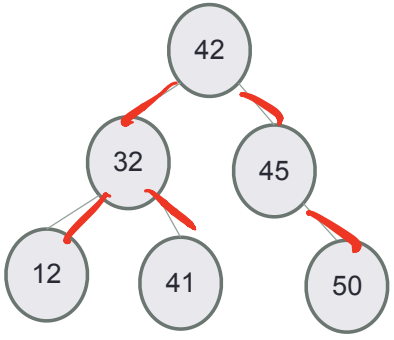


- Given a BST of height H and N nodes, what is the worst case complexity of searching for a key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)



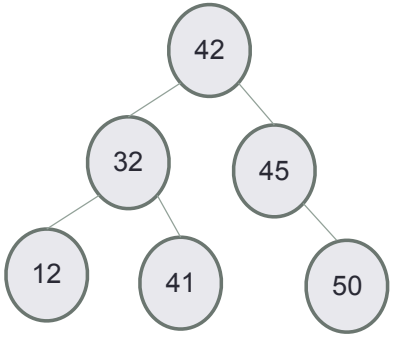
Worst case Big-O of insert

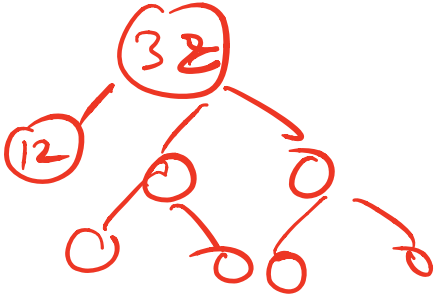


- Given a BST of height H and N nodes, what is the worst case complexity of inserting a key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

Worst case Big-O of min/max

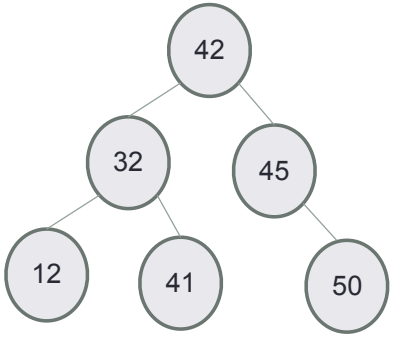




- Given a BST of height H and N nodes, what is the worst case complexity of finding the minimum or maximum key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

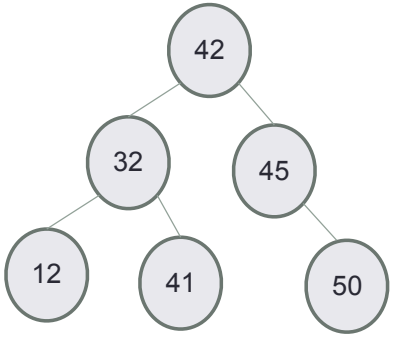
Worst case Big-O of predecessor/successor



- Given a BST of height H and N nodes, what is the worst case complexity of finding the minimum or maximum key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

Worst case Big-O of delete



- Given a BST of height H and N nodes, what is the worst case complexity of deleting the key (assume no duplicates)?

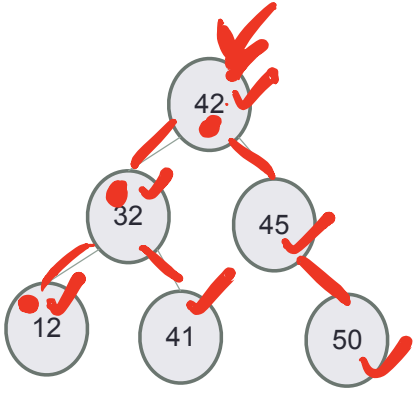
A. O(1)

B. O(log N)

C. O(H)

D. O(log H)

Big O of traversals



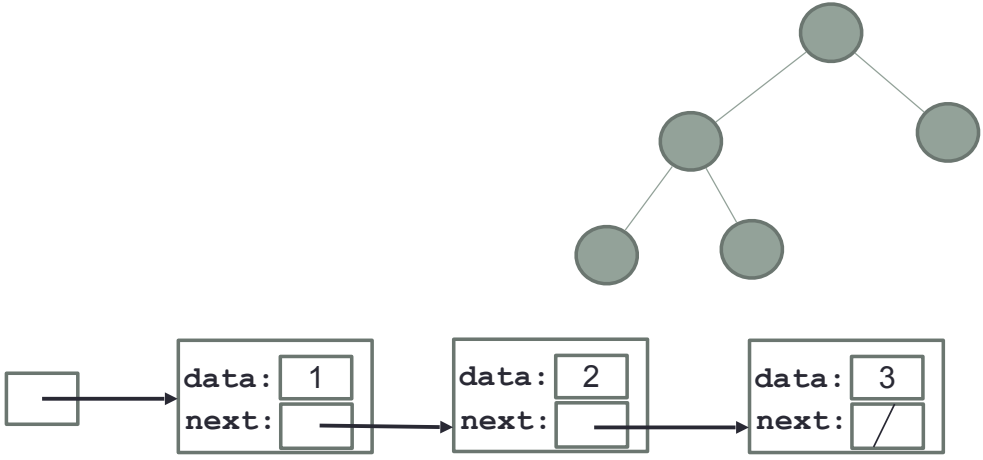
In Order:
Pre Order:
Post Order:

O(N)

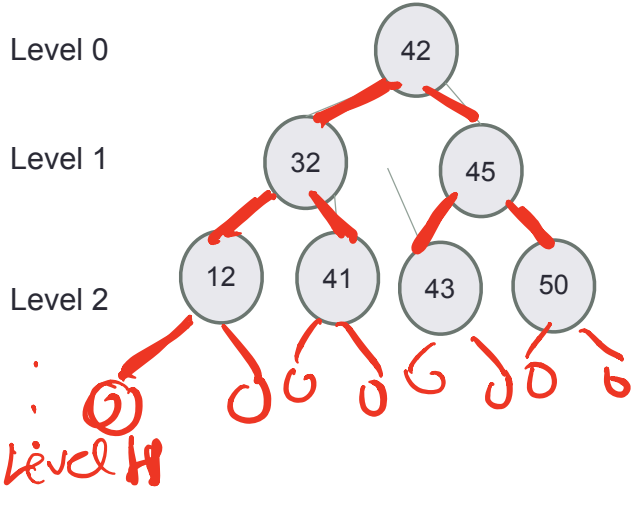
Worst case analysis

Are binary search trees *really* faster than linked lists for finding elements?

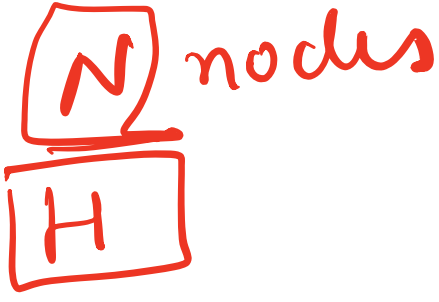
- A. Yes
- B. No



Completely filled binary tree



Nodes at each level have exactly two children, except the nodes at the last level



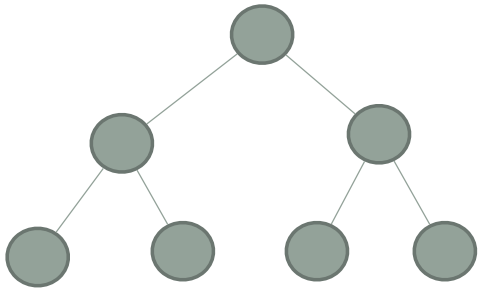
Relating H (height) and N (#nodes)

find is O(H), we want to find a f(N) = H

$1 + 2 + 3 + \cdots 2^7 = 255$

$2^8 - 1$

Level	no. of nodes
0	1
1	2
3	4
⋮	⋮
H	2^H



How many nodes are on level L in a completely filled binary search tree?

A. 2

B. L

C. 2*L

D. 2^L

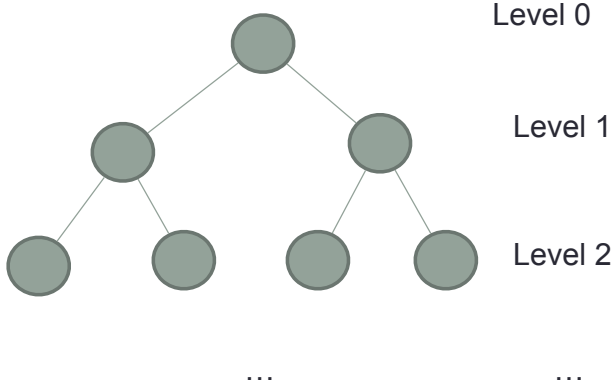
$\sum_{i=0}^{H} 2^i$ | $1 + 2 + 4 + 8 + \cdots 2^H = N$

$= 2^{H+1} - 1 = N$

$H = \log(N+1) - 1$

Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H

$H = \log(N+1) - 1$



Finally, what is the height (exactly) of the tree in terms of N?

$O(\log N)$

Balanced trees

- Balanced trees by definition have a height of O(log N)
- A completely filled tree is one example of a balanced tree
- Other Balanced BSTs include AVL trees, red black trees and so on
- Visualize operations on an AVL tree: https://visualgo.net/bn/bst

Summary of operations

Operation	Sorted Array	Binary Search Tree	Linked List
Min	O(1)	O(log N)	O(N)
Max	O(1)	O(log N)	O(N)
Median	O(1)	—	O(N)
Successor	O(1)	O(log N)	O(N)
Predecessor	O(1)	O(log N)	O(N)
Search	O(log N)	O(log N)	O(N)
Insert	O(N)	O(log N)	O(1)
Delete	O(N)	O(log N)	O(1)
Print	O(N)	O(N)	

Balanced

(unsorted)

Binary search

(if you know which node to delete)

CHANGING GEARS: C++STL

- The C++ Standard Template Library is a very handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

C++ STL container classes

```
                        array
                       vector
                  forward_list
                         list
                        stack
                        queue
                  priority_queue
                          set
multiset (non unique keys)
                        deque
                  unordered_set
                          map
                  unordered_map
                       multimap
                         bitset
```