

BINARY SEARCH TREES RUNNING TIME

Problem Solving with Computers-II

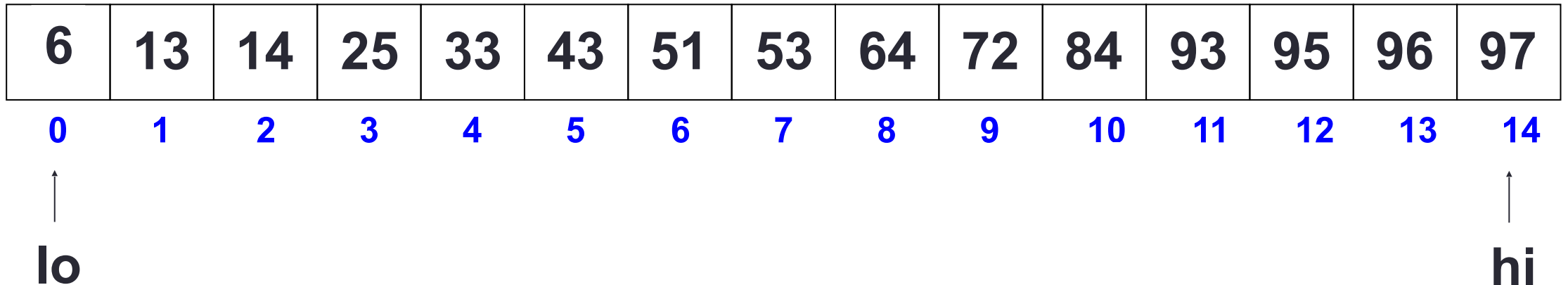
The image shows the C++ logo in blue, followed by a snippet of C++ code in a monospaced font. The code is:

```
#include <iostream>
using namespace std;
int main(){
    cout<<"Hola Facebook!n";
    return 0;
}
```

Best case, worst case, average case running times

Operations on sorted arrays

- Min :
- Max:
- Median:
- Successor:
- Predecessor:
- Search:
- Insert :
- Delete:



Worst case analysis of binary search

```
bool binarySearch(int arr[], int element, int N){
//Precondition: input array arr is sorted in ascending order
    int begin = 0;
    int end = N-1;
    int mid;
    while (begin <= end){
        mid = (end + begin)/2;
        if(arr[mid]==element){
            return true;
        }else if (arr[mid]< element){
            begin = mid + 1;
        }else{
            end = mid - 1;
        }
    }
    return false;
}
```

Binary Search Trees

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

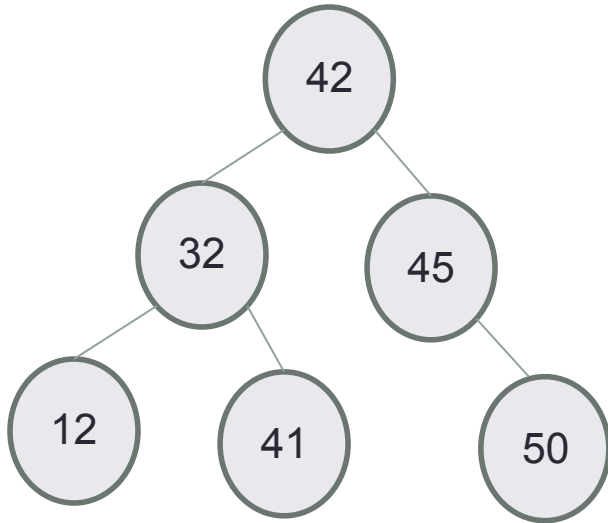
Height of the tree



- 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.

BSTs of different heights are possible with the same set of keys
Examples for keys: 12, 32, 41, 42, 45

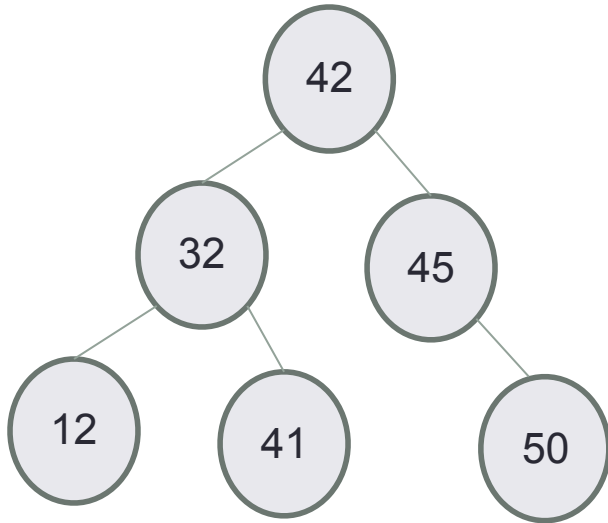
Worst case Big-O of search



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

- A. $O(1)$
- B. $O(\log H)$
- C. $O(H)$
- D. $O(H \cdot \log H)$
- E. $O(N)$

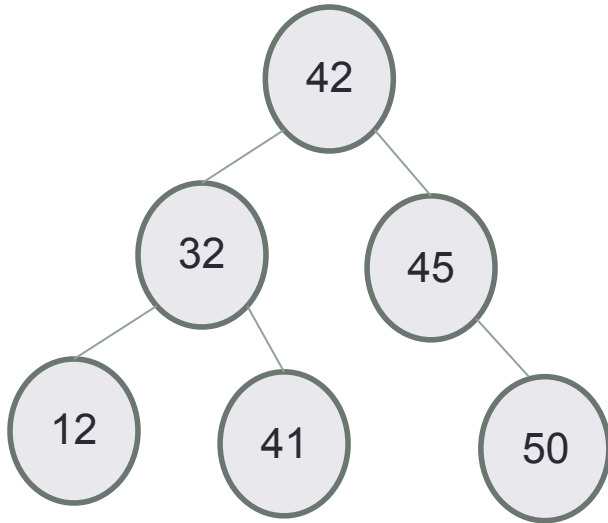
Worst case Big-O of insert



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

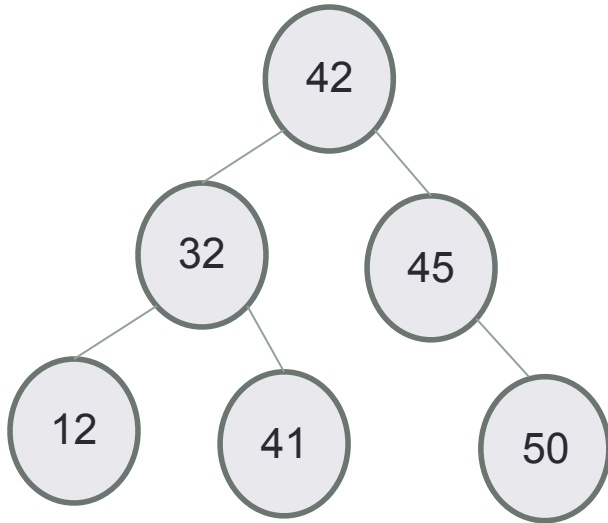
- A. $O(1)$
- B. $O(\log H)$
- C. $O(H)$
- D. $O(H \cdot \log H)$
- E. $O(N)$

Worst case Big-O of min/max



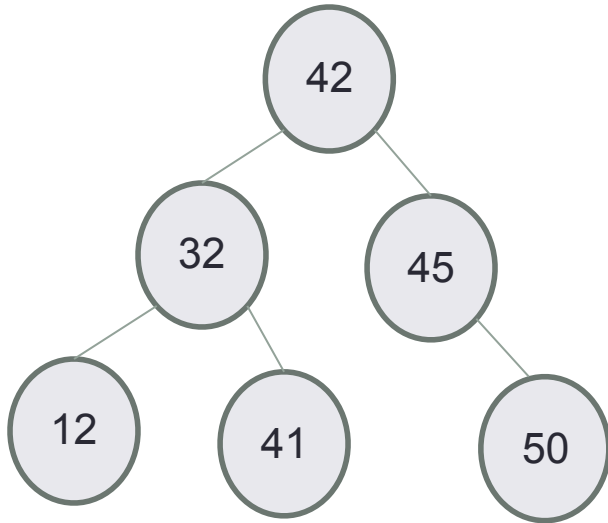
- Given a BST of height H and N nodes, what is the worst case complexity of finding the minimum or maximum key?
- $O(1)$
 - $O(\log H)$
 - $O(H)$
 - $O(H \cdot \log H)$
 - $O(N)$

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 predecessor or successor key?
- $O(1)$
 - $O(\log H)$
 - $O(H)$
 - $O(H \cdot \log H)$
 - $O(N)$

Worst case Big-O of delete

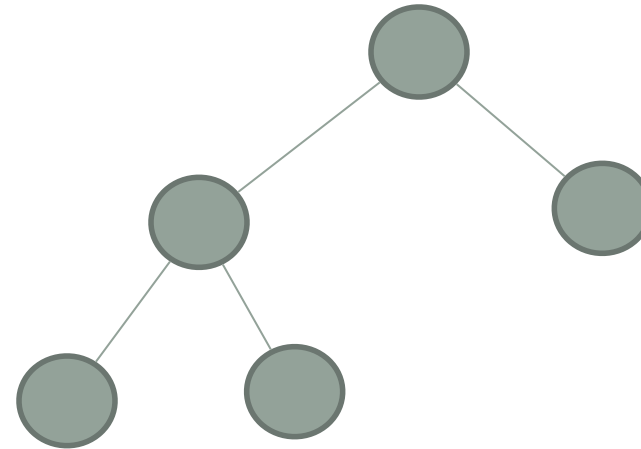


- Given a BST of height H and N nodes, what is the worst case complexity of deleting the key (assume no duplicates)?
 - A. $O(1)$
 - B. $O(\log H)$
 - C. $O(H)$
 - D. $O(H \cdot \log H)$
 - E. $O(N)$

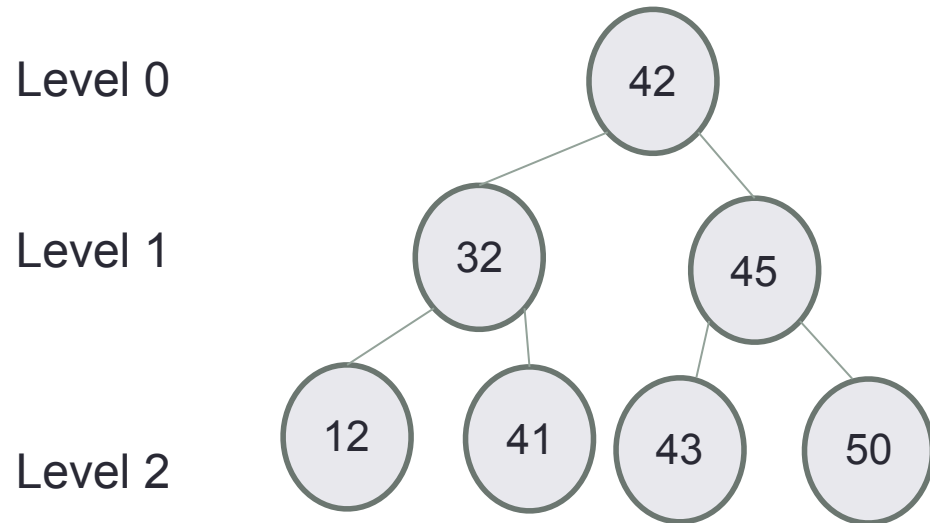
Worst case analysis

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

- A. Yes
- B. No

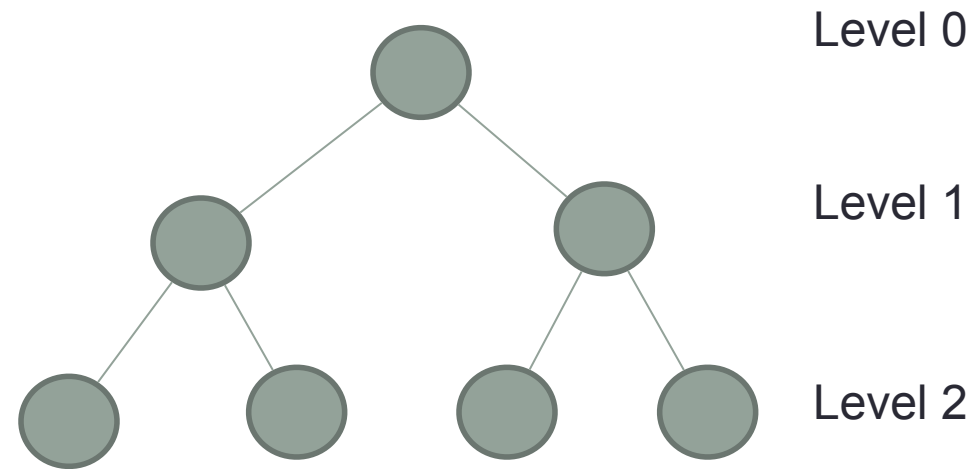


Completely filled binary tree



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 $f(N) = H$



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

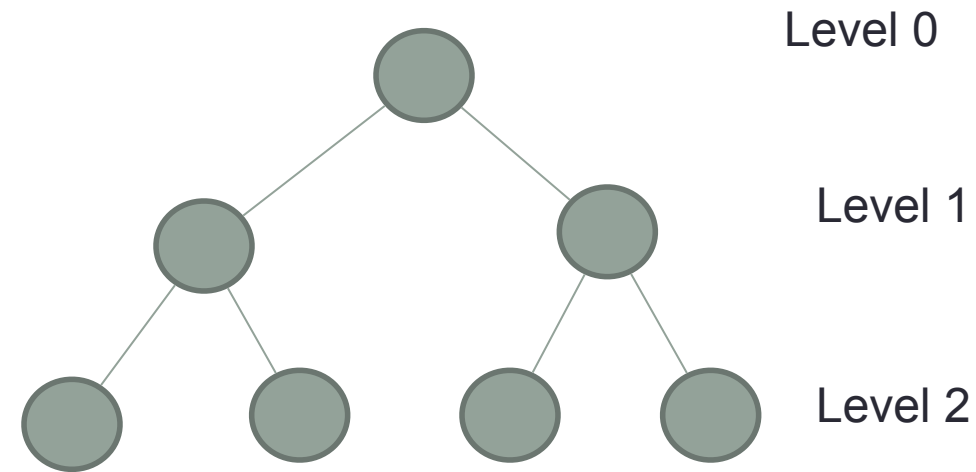
A.2

B.L

C. 2^L

D. 2^L

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

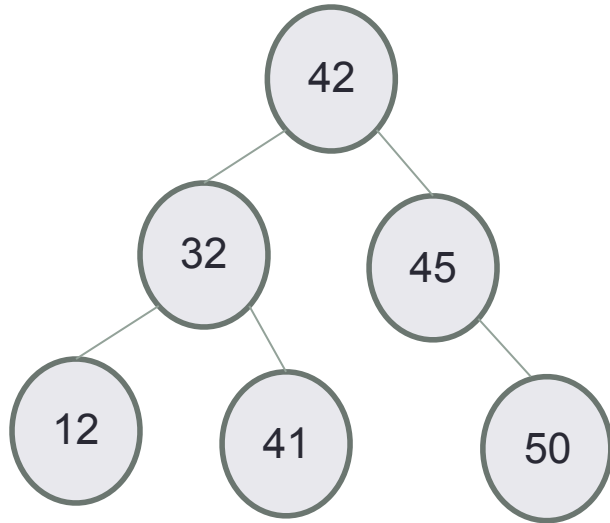


Finally, what is the height (exactly) of the tree in terms of N ?
...

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>

Big O of traversals



In Order:

Pre Order:

Post Order:

Summary of operations

Operation	Sorted Array	Binary Search Tree	Linked List
Min			
Max			
Median			
Successor			
Predecessor			
Search			
Insert			
Delete			