SORTED ARRAYS REVISTED
BINARY SEARCH TREES

Problem Solving with Computers-II

How much has the mentor program helped you feel supported in your coursework?

A. None at all
B. Mildly helpful
C. Moderately
D. High level of help
E. N/A : Did not sufficiently interact with the mentors

How much has the mentor program helped you better understand the course content?

A. None at all
B. Mildly helpful
C. Moderately
D. High level of help
E. N/A : Did not sufficiently interact with the mentors

How much has the mentor program helped you improve your debugging skills?

A. None at all
B. Mildly helpful
C. Moderately
D. High level of help
E. N/A : Did not sufficiently interact with the mentors

How much has the mentor program helped you feel part of a community of peers?

A. None at all
B. Mildly helpful
C. Moderately
D. High level of help
E. N/A : Did not sufficiently interact with the mentors

How much do you value the open lab hours offered in this course?

A. None at all
B. Mildly
C. Moderately
D. It’s a great resource
E. N/A : I have never tried to go

Operations on sorted arrays

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

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo														↑ hi

Binary Search

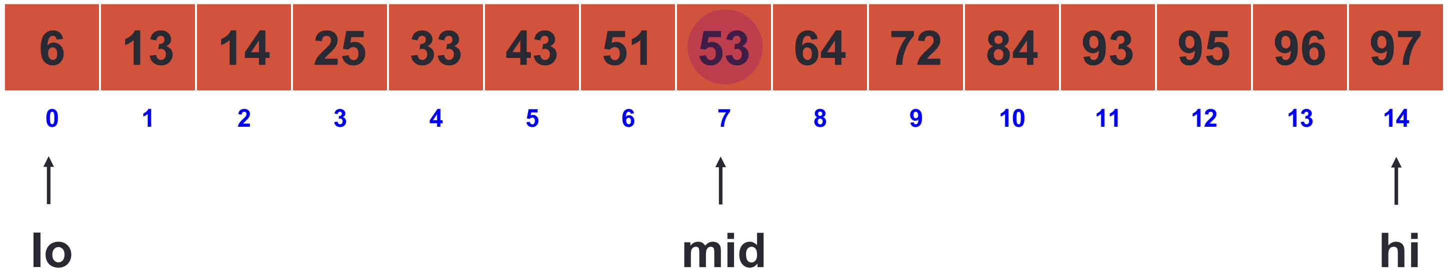
- Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Invariant. Algorithm maintains `a[lo]` ≤ `value` ≤ `a[hi]`.

- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑														↑
lo														hi

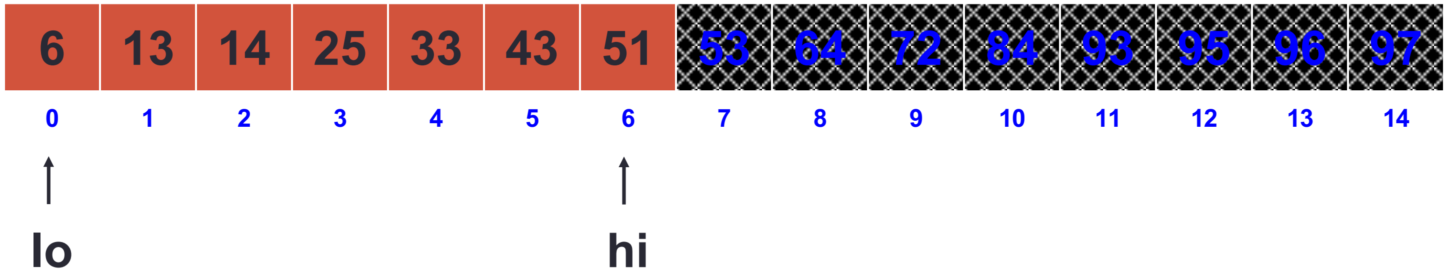
Binary Search

- Ex. Binary search for 33.



Binary Search

- Ex. Binary search for 33.



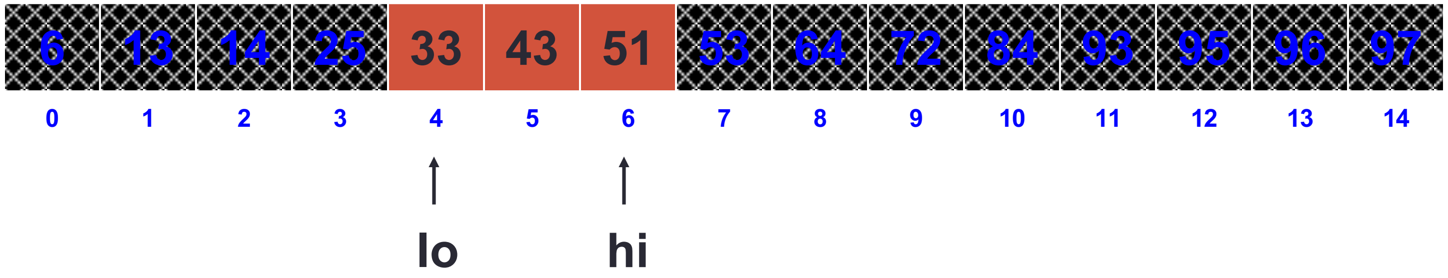
Binary Search

- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo			↑ mid			↑ hi								

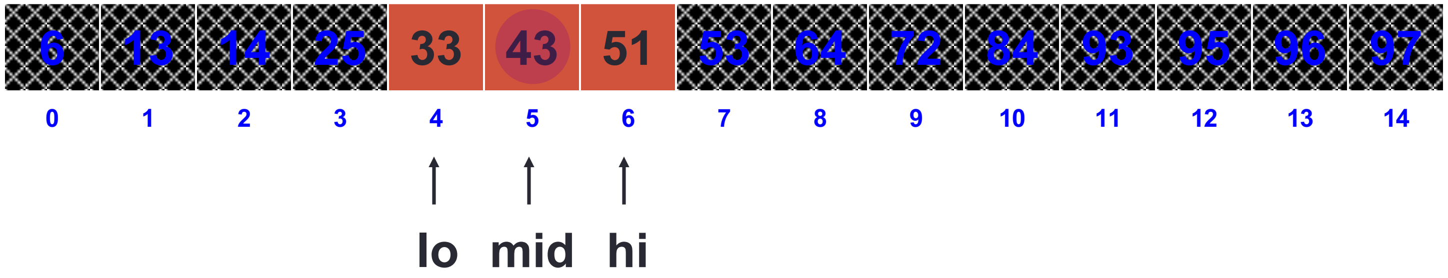
Binary Search

- Ex. Binary search for 33.



Binary Search

- Ex. Binary search for 33.



Binary Search

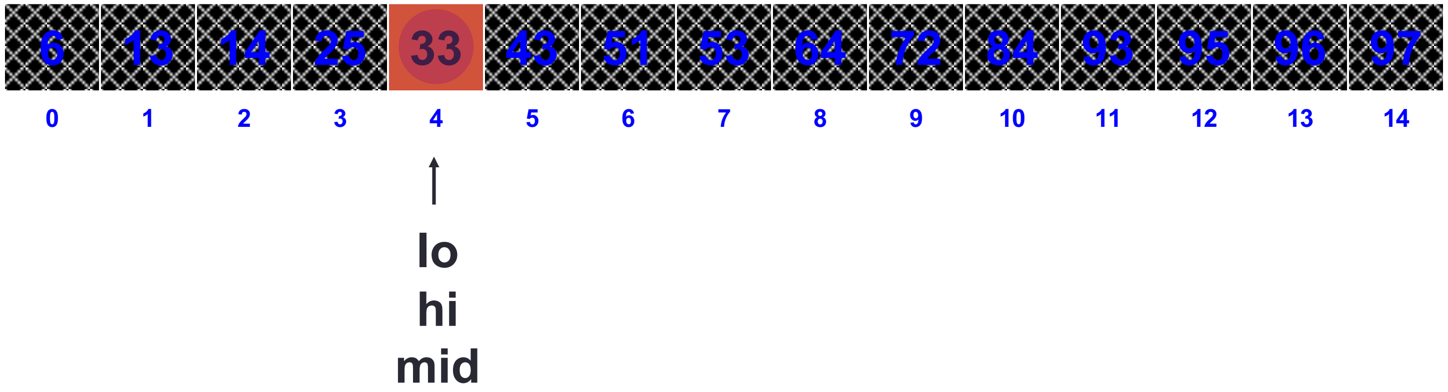
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo
hi

Binary Search

- Ex. Binary search for 33.



Binary Search

- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

↑ (at index 4)

lo
hi
mid

Binary Search Run Time

- What is run time complexity of Binary Search on an array of size N?

A. O(1)

B. O(log N)

C. O(N)

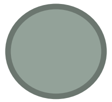
D. O(N*logN)

E. O(2^N)

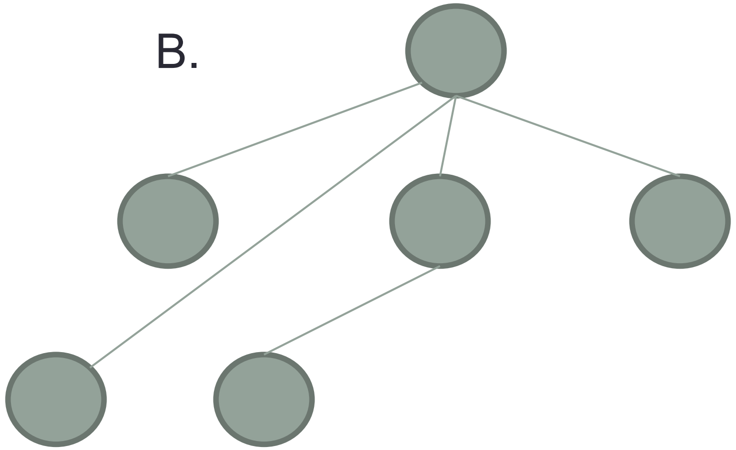
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo														↑ hi

Which of the following is/are a tree?

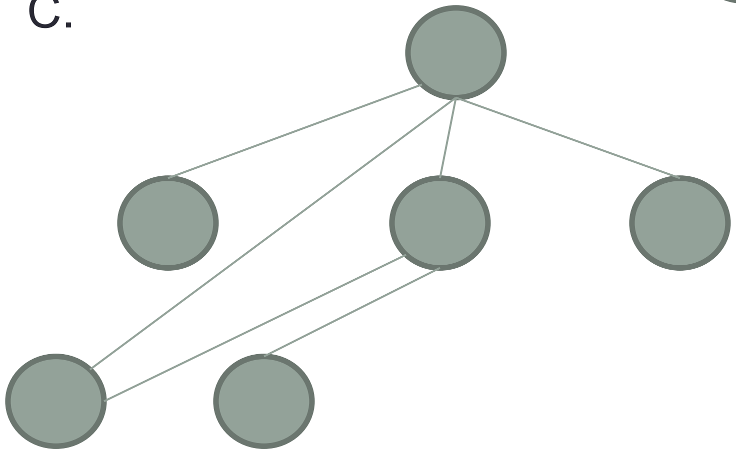
A.



B.

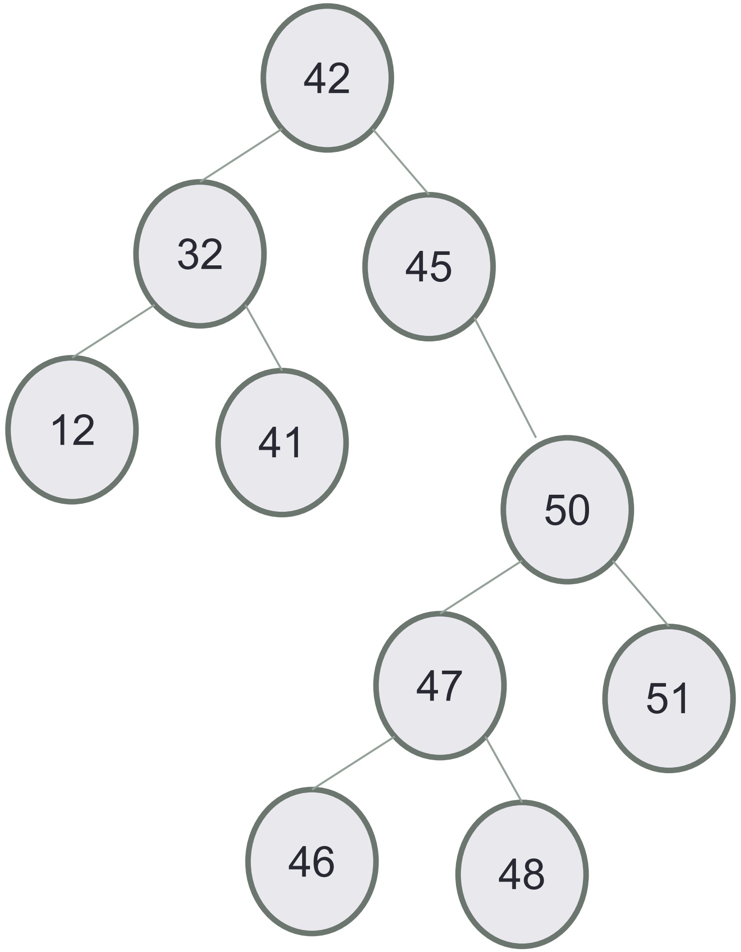


C.



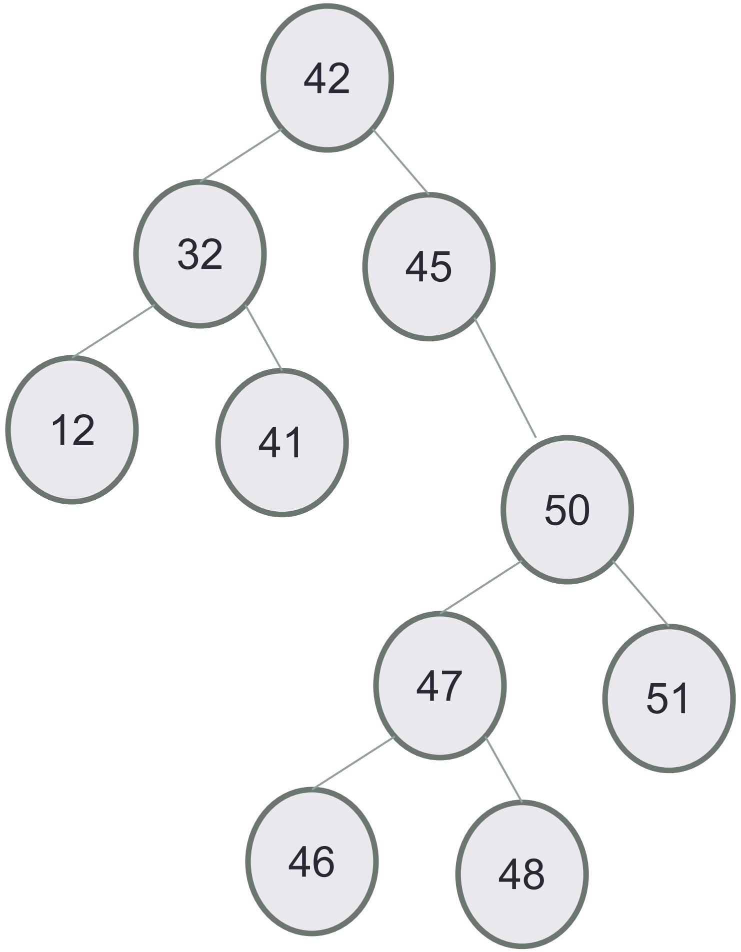
D. A & B
E. All of A-C

Lab04: Binary Search Tree – What is it?



What are the numbers in the nodes?

Binary Search Tree – What is it?

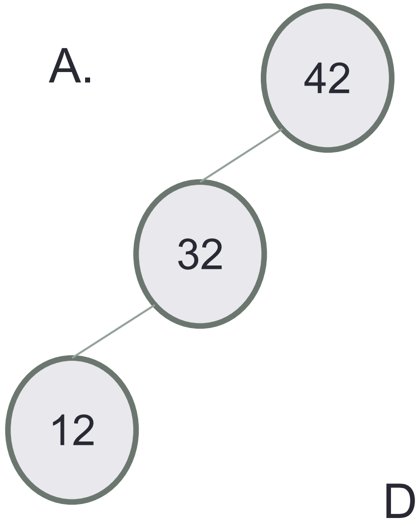


For any node,
Keys in node's left subtree <= Node key
Node key < Keys in node's right subtree

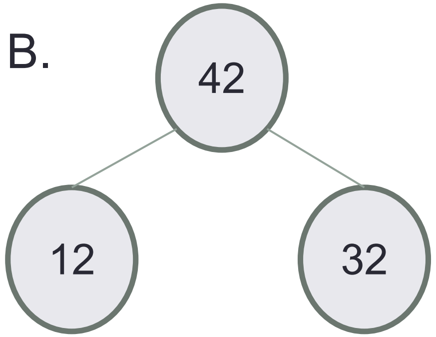
Do the keys have to be integers?

Which of the following is/are a binary search tree?

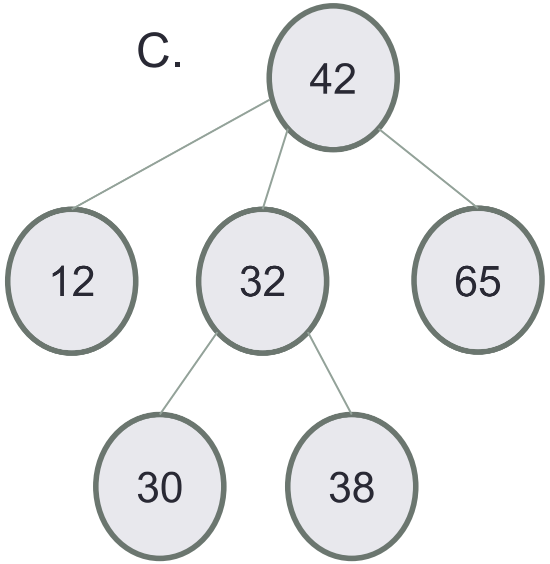
A.



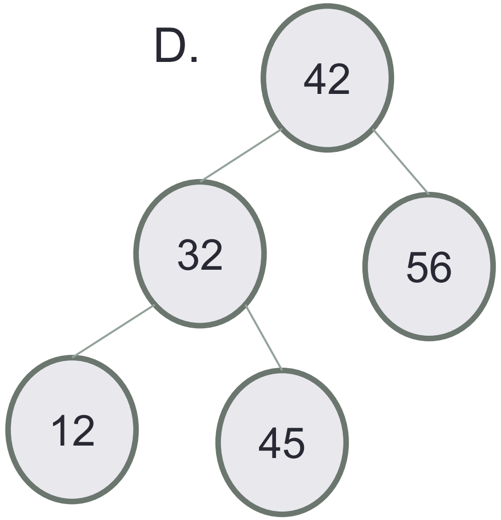
B.



C.



D.



E. More than one of these

A node in a BST

```
class BSTNode {

public:
   BSTNode* left;
   BSTNode* right;
   BSTNode* parent;
   int const data;

   BSTNode( const int & d ) : data(d) {
      left = right = parent = 0;
   }
};
```

Binary Search Trees

- What are the operations supported?
- What are the running times of these operations?
- How do you implement the BST i.e. operations supported by it?

Binary Search Trees

- What is it good for?
  - If it satisfies a special property i.e. Balanced, you can think of it as a dynamic version of the sorted array

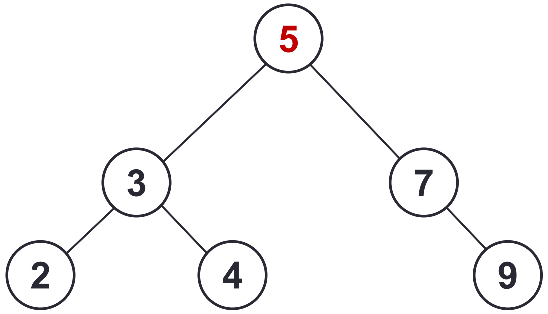
Traversing a Binary Search Tree

- **Inorder** tree walk:
  - Root is printed between the values of its left and right subtrees:

    left, root, right ➔ keys are printed in sorted order

- **Preorder** tree walk: root printed first: **root, left, right**

- **Postorder** tree walk: root printed last: **left, right, root**



Inorder: 2 3 4 **5** 7 9

Preorder: **5** 3 2 4 7 9

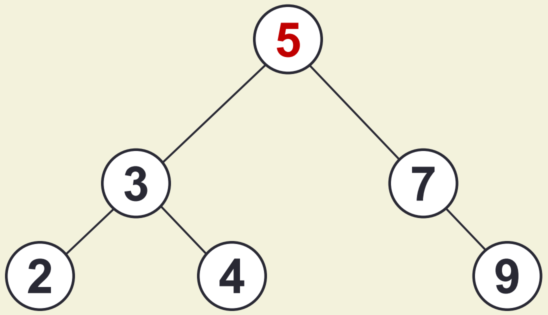
Postorder: 2 4 3 9 7 **5**

Traversing a Binary Search Tree

Alg: **INORDER-TREE-WALK(x)**

```
if x ≠ NIL
   then INORDER-TREE-WALK ( left [x] )
        print key [x]
        INORDER-TREE-WALK ( right [x] )
```

E.g.:

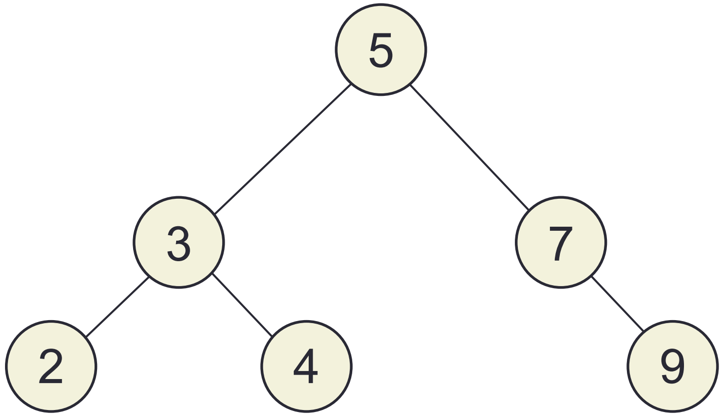


Output: 2 3 4 **5** 7 9

- Running time:
  - **Θ(N)**, where N is the size of the tree rooted at x

Searching for a Key

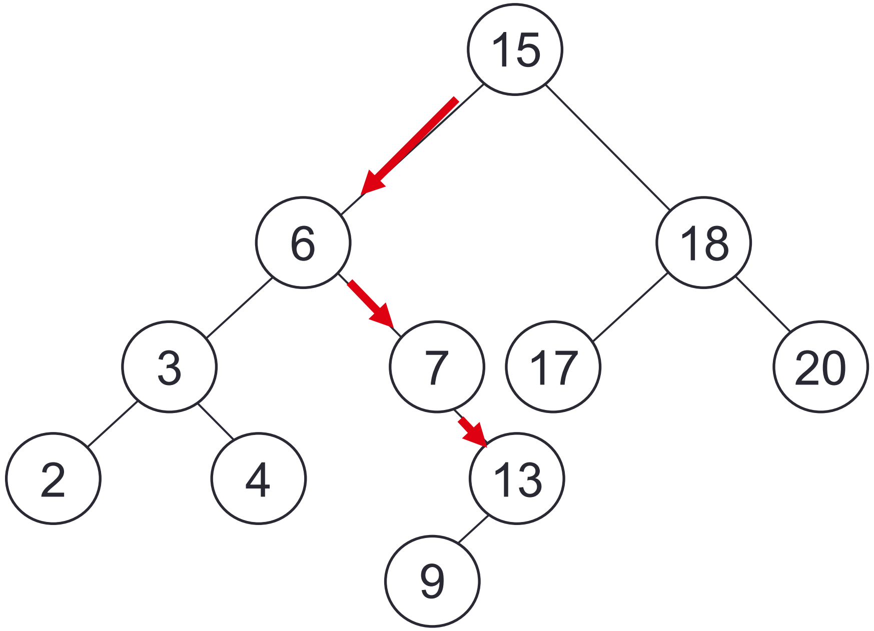
- Given a pointer to the root of a tree and a key **k**:
  - Return a pointer to a node with key k if one exists
  - Otherwise return NIL



- Start at the root; trace down a path

  by comparing **k** with the key of the current node x:
  - If the keys are equal: we have found the key
  - If **k** < key[x] search in the left subtree of x
  - If **k** > key[x] search in the right subtree of x

Example: TREE-SEARCH

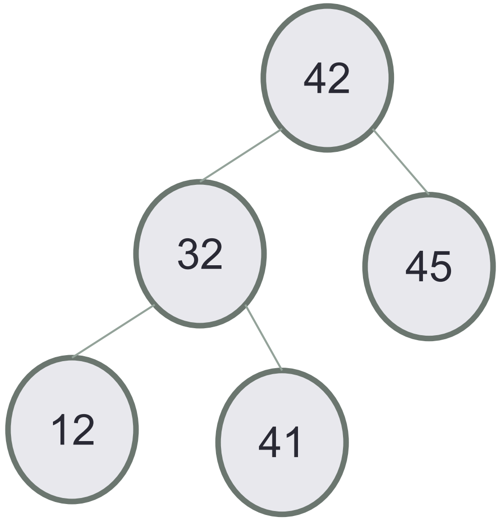


- Search for key 13:

15 → 6 → 7 → 13

Tree Search example

Search for 41.
Now search for 43.



Height of a node: the height of a node is the number of edges on the longest path from the node to a leaf
Height of a tree: the height of the root of the tree
Height of this tree is 2.

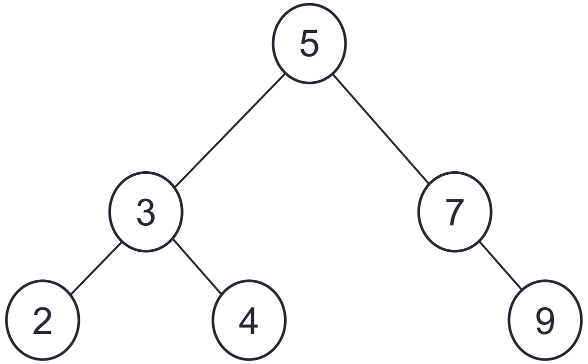
Searching for a Key

Alg: **TREE-SEARCH(x, k)**

```
if x = NIL or k = key [x]
    then return x
if k < key [x]
    then return TREE-SEARCH(left [x], k )
    else return TREE-SEARCH(right [x], k )
```

Running Time:

$O(H)$, H – the height of the tree



Finding the Minimum in a Binary Search Tree

Goal: find the minimum value in a BST

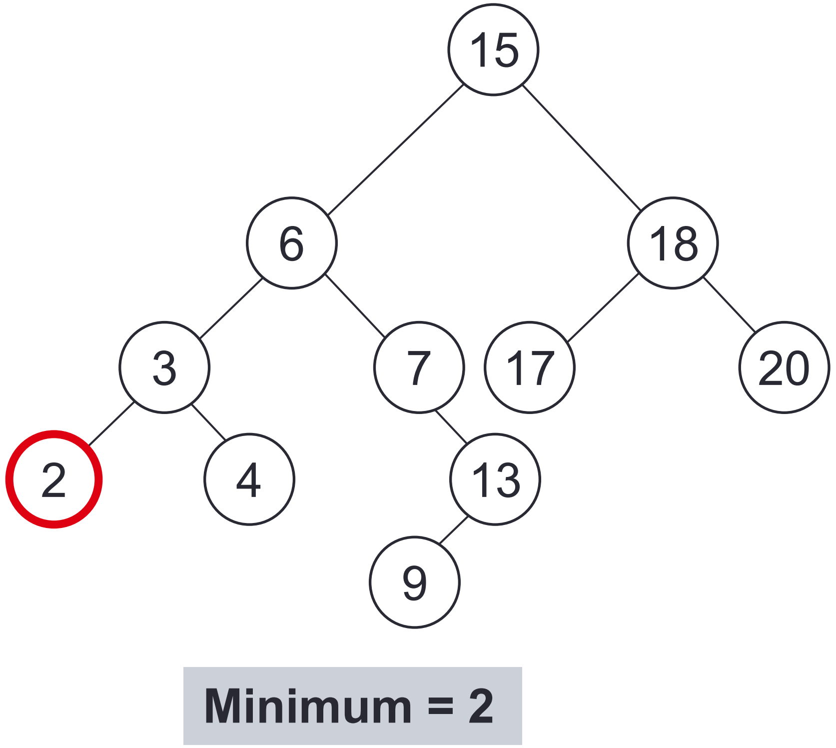
Following left child pointers from the root, until a NIL is encountered

Alg: TREE-MINIMUM(x)

1. **while** left [x] ≠ NIL
2. **do** x ← left [x]
3. **return** x

Running time

O(H), H – height of tree



Finding the Maximum in a Binary Search Tree

Goal: find the maximum value in a BST

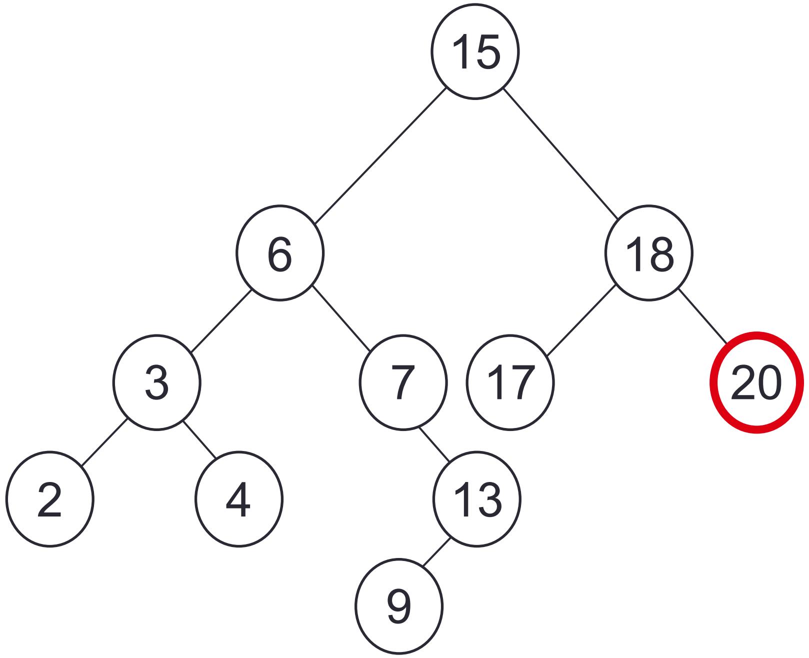
Following right child pointers from the root, until a NIL is encountered

Alg: TREE-MAXIMUM(x)

1. **while** right [x] $\neq$ NIL
2. **do** x $\leftarrow$ right [x]
3. **return** x

Running time

O(h), h – height of tree

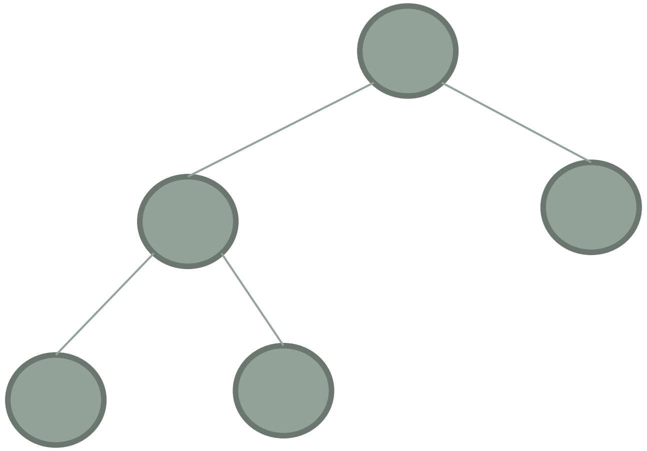


Maximum = 20

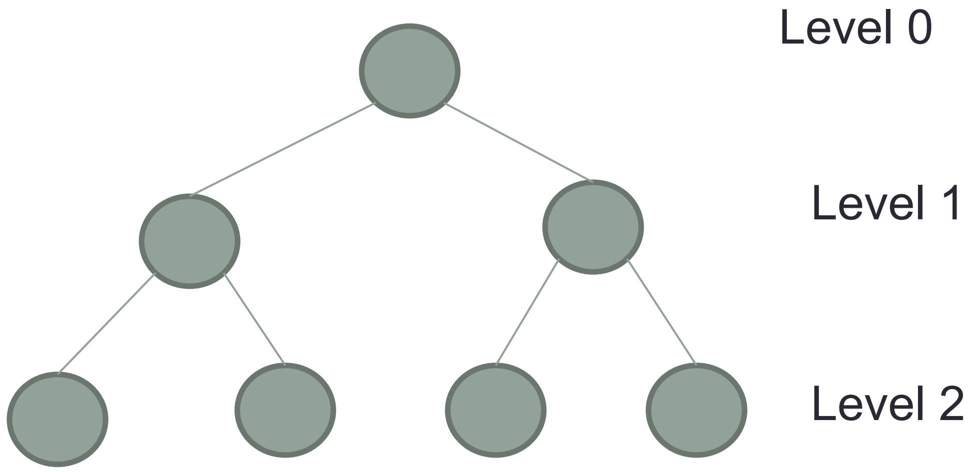
How fast is BST search algorithm?

How long does it take to find an element in the tree in terms of the tree's height, H?

O(H)….. But we really want to describe this in terms of the number of nodes in the tree



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



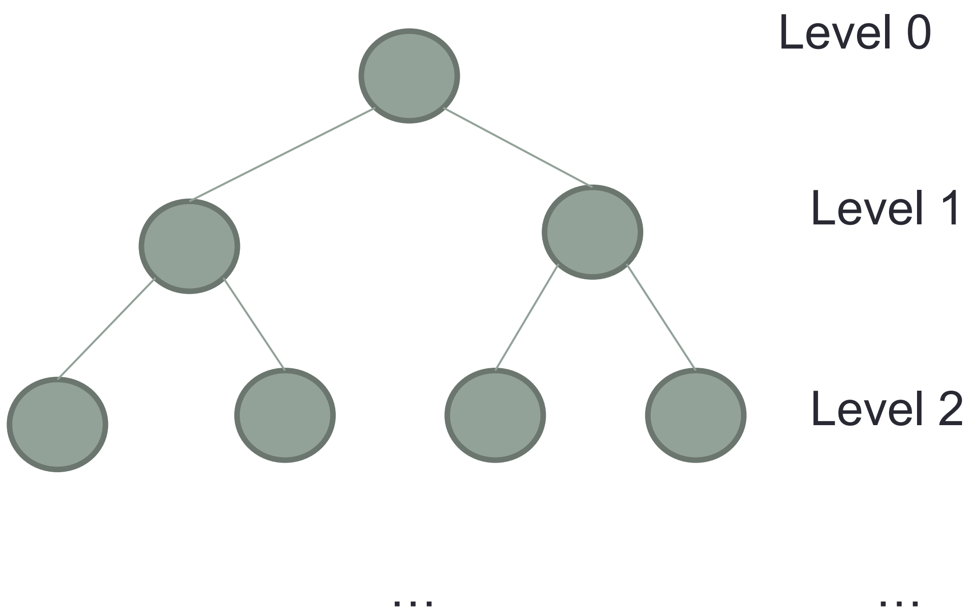
...   ...

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

$$N = \sum_{L=0}^{H} 2^L = 2^{H+1} - 1$$



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

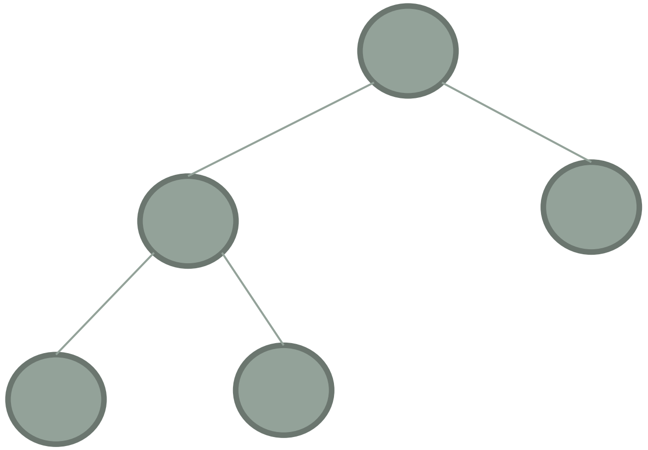
$$H = \log_2(N+1) - 1$$

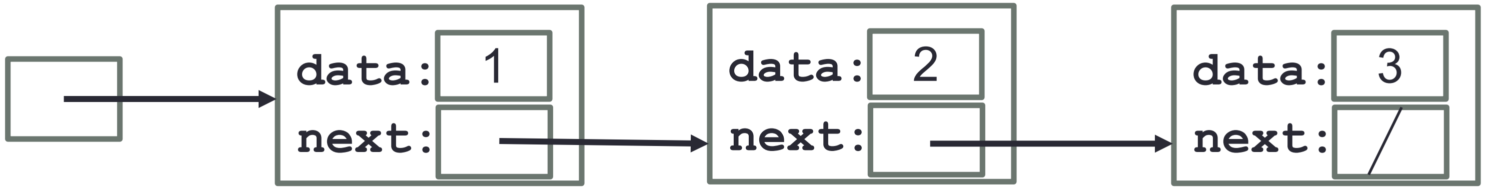
And since we knew finding a node was O(H), we now know it is $O(\log_2 N)$

Worst case analysis

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

- A. Yes
- B. No



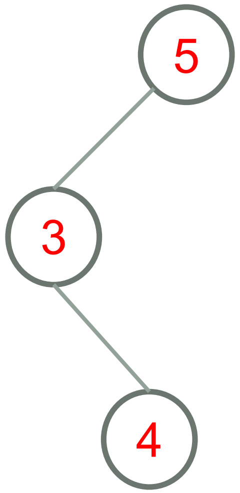


Average case analysis of a “successful” find

Given a BST having N nodes x_1, .. x_N, such that key(x_i) = k_i

How many compares to locate a key in the BST?

1. Worst case:
2. Best case:
3. Average case:



Here is the result! Proof is a bit involved but if you are interested in the proof, come to office hours

$D_{avg}(N)$	Average #comparisons to find a single item in any BST with N nodes

$$D_{avg}(N) \approx 1.386\ \log_2 N$$

Conclusion: The average time to find an element in a BST with no restrictions on shape is Θ(log N).