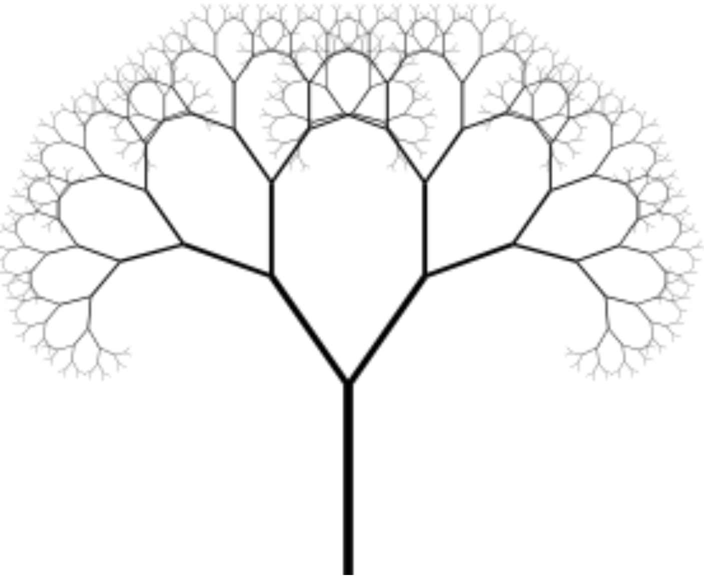
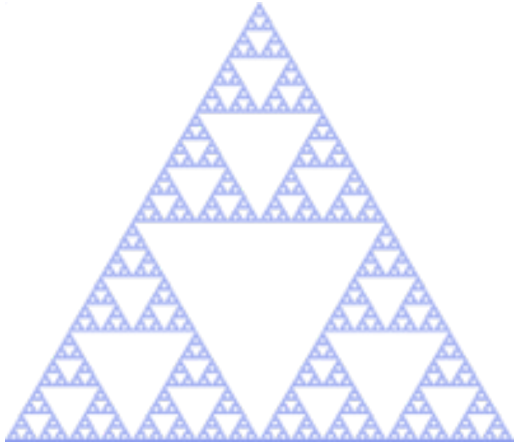
RECURSION

Problem Solving with Computers-II

Recursion

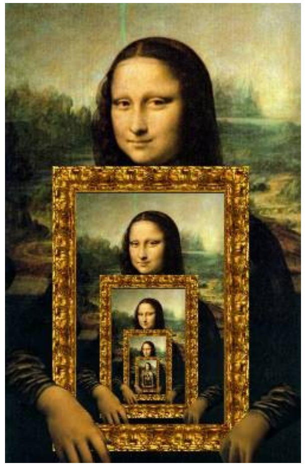


Fractal Tree



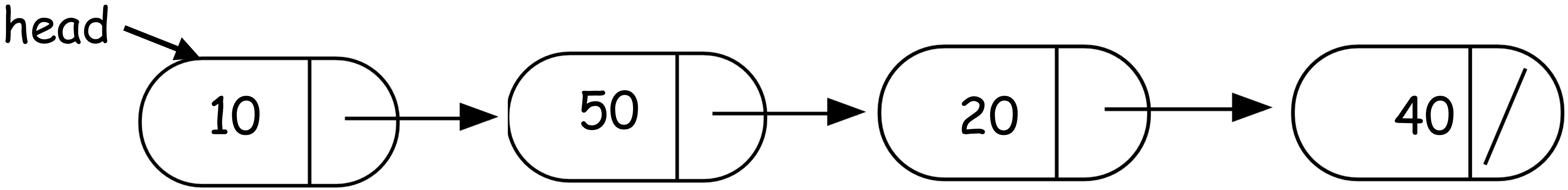
Sierpinski triangle

Koch's snowflake



Which of the following methods of class LinkedList CANNOT be implemented using recursion?

A. Finding the sum of all the values
B. Printing all the values
C. Deleting all the nodes in a linked list
D. Searching for a value
E. All the above can be implemented using recursion



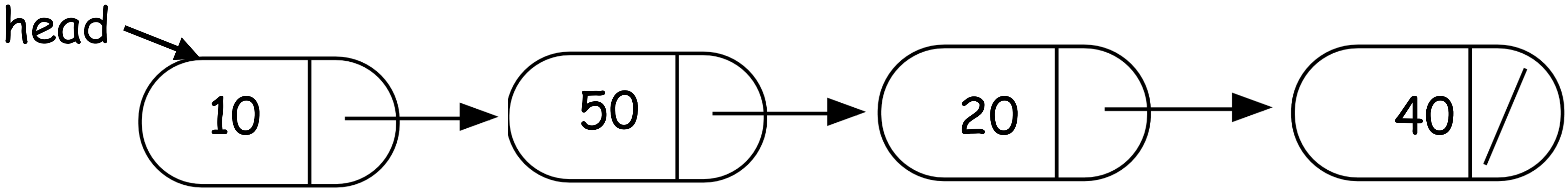

```
int IntList::sum(){

     //Return the sum of all elements in a linked list
  }
```

Helper functions

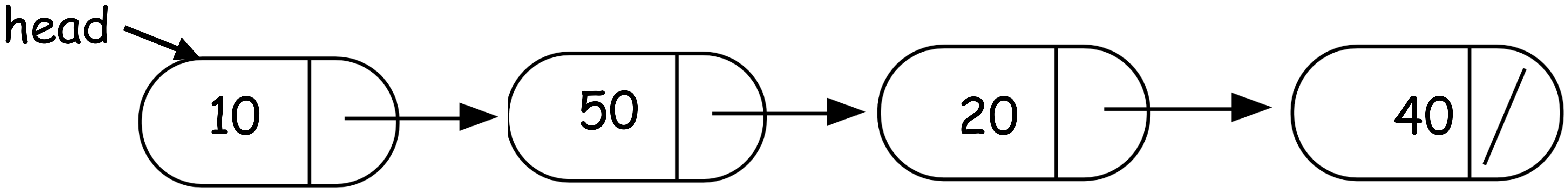
- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion
- Usually the helper function is private

For example

```
Int IntList::sum(){

   return sum(head);
    //helper function that performs the recursion.

}
```




```
int IntList::sum(Node* p){



}
```




```
bool IntList::clear(Node* p){



 }
```

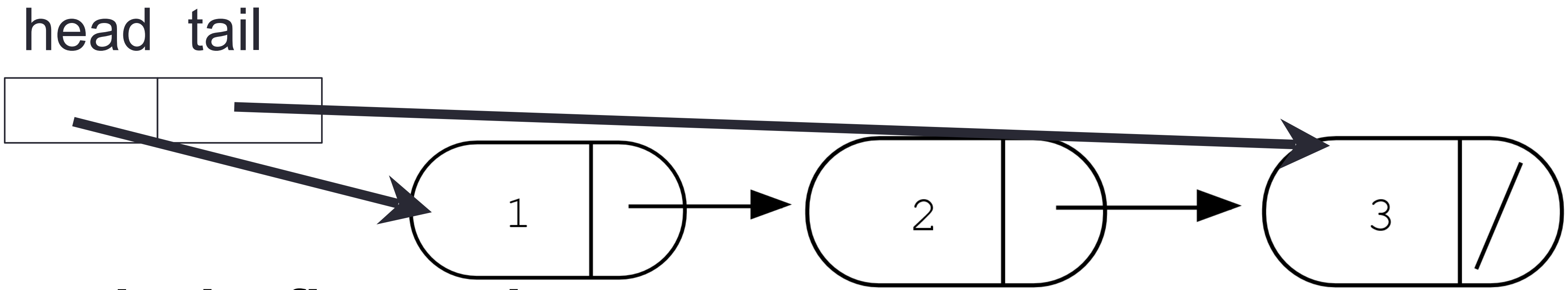
Concept Question

```
LinkedList::~LinkedList(){
    delete head;
}
```

```
class Node {
    public:
      int info;
      Node *next;
};
```

Which of the following objects are deleted when the destructor of Linked-list is called?

(A)



(B): only the first node

(C): A and B

(D): All the nodes of the linked list

(E): A and D

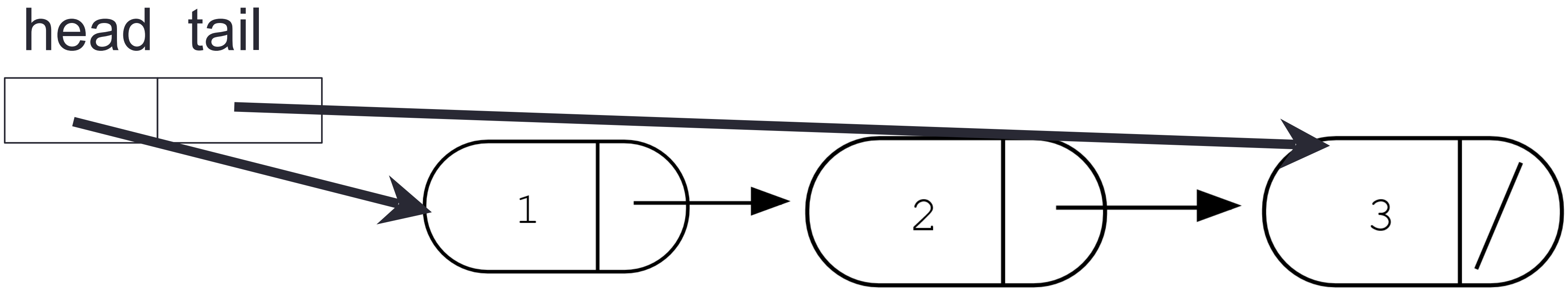
Concept question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```

Which of the following objects are deleted when the destructor of Linked-list is called?

(A)



(B): All the nodes in the linked-list

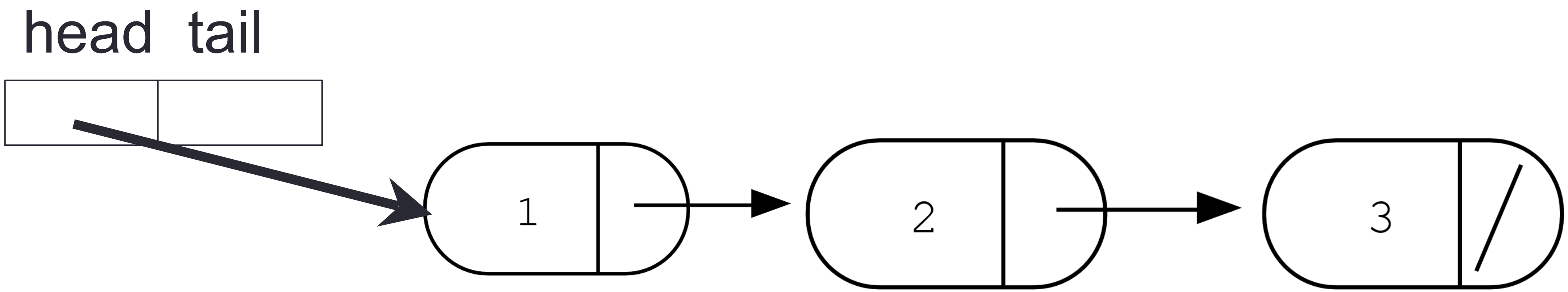
(C): A and B

(D): Program crashes with a segmentation fault

(E): None of the above

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```



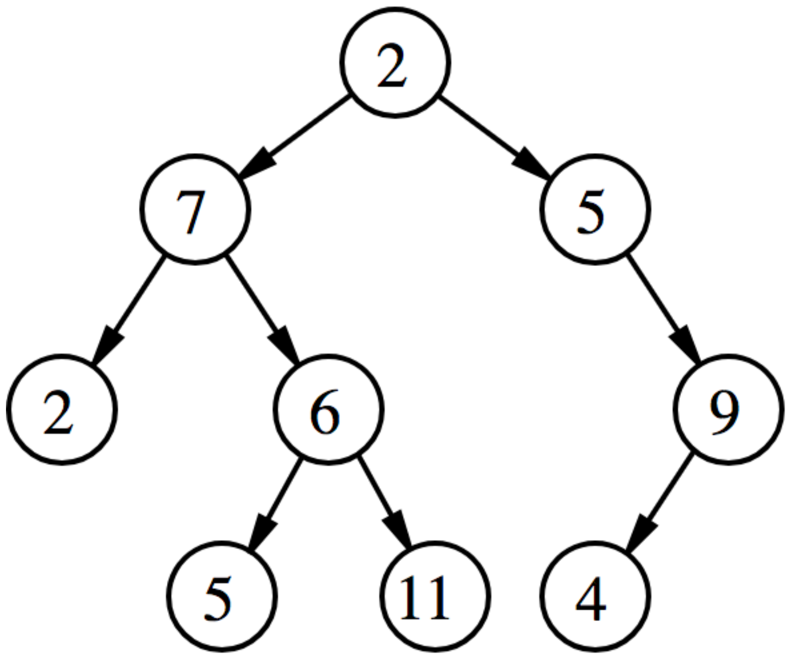
Binary Search

- 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

Trees



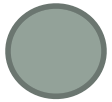
A tree has following general properties:

- One node is distinguished as a **root**;
- Every node (exclude a root) is connected by a directed edge *from* exactly one other node;

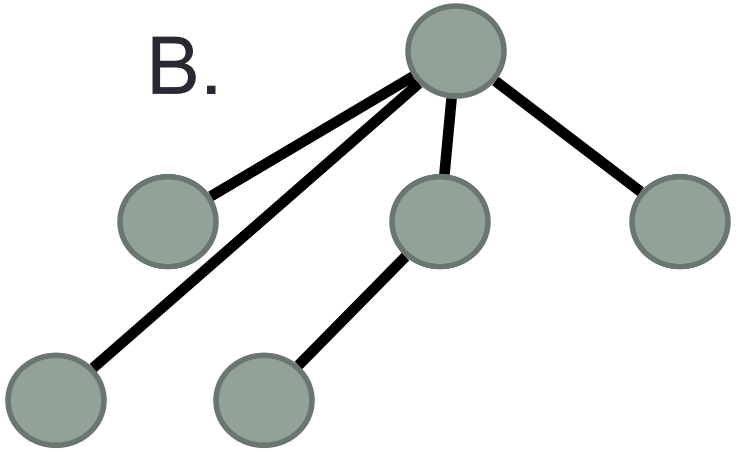
  A direction is: *parent -> children*
- *Leaf node: Node that has no children*

Which of the following is/are a tree?

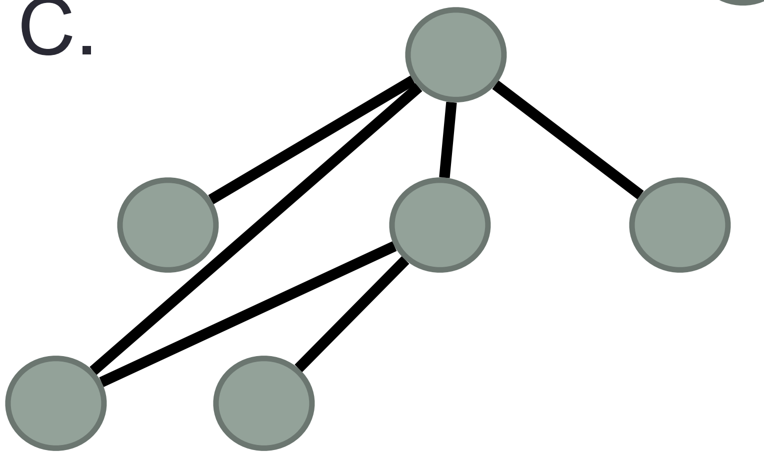
A.



B.



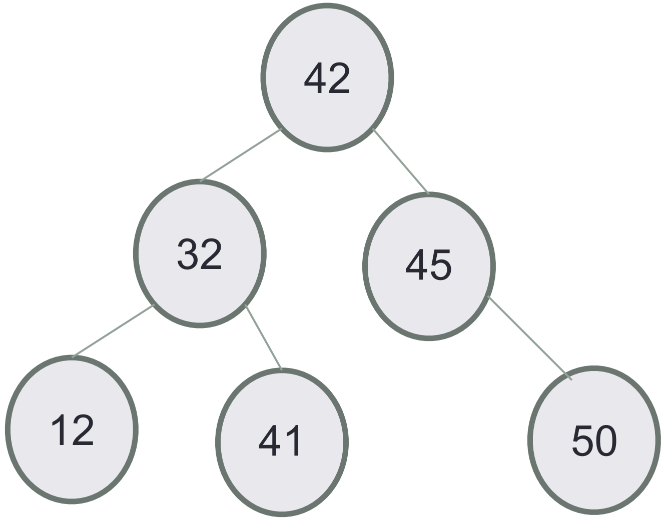
C.



D. A & B

E. All of A-C

Binary Search Tree – What is it?



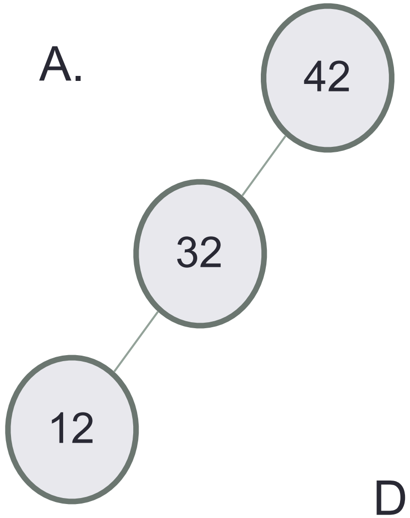
- Each node:
  - stores a key (k)
  - has a pointer to left child, right child and parent (optional)
  - Satisfies the Search Tree Property

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

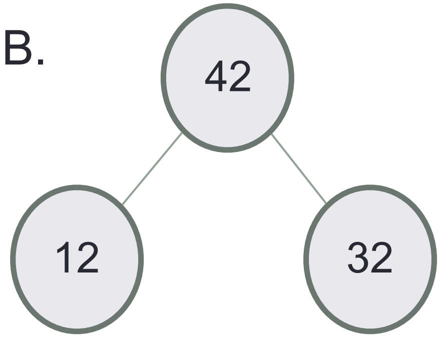
Do the keys have to be integers?

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

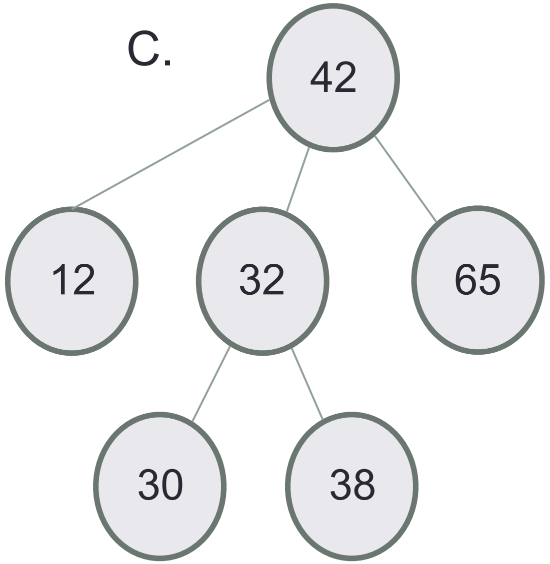
A.



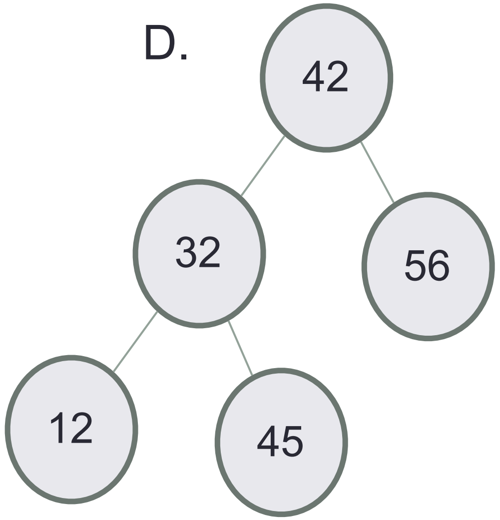
B.



C.

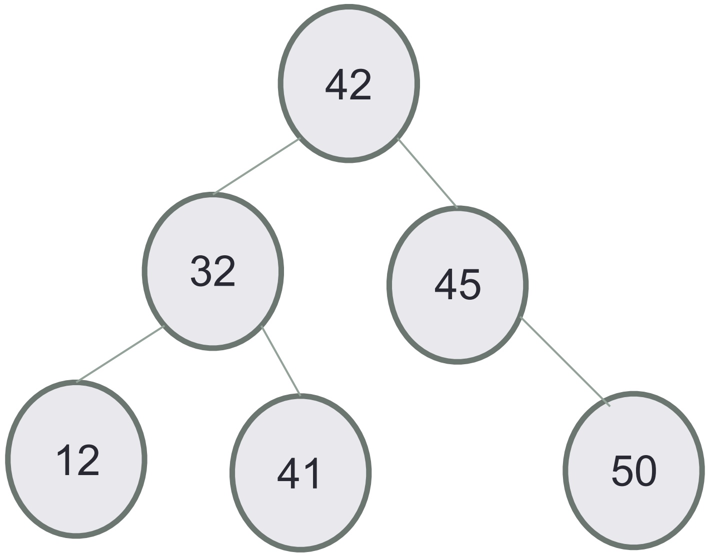


D.



E. More than one of these

BSTs allow efficient search!



- 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

Search for 41, then search for 53