Binary Search Trees

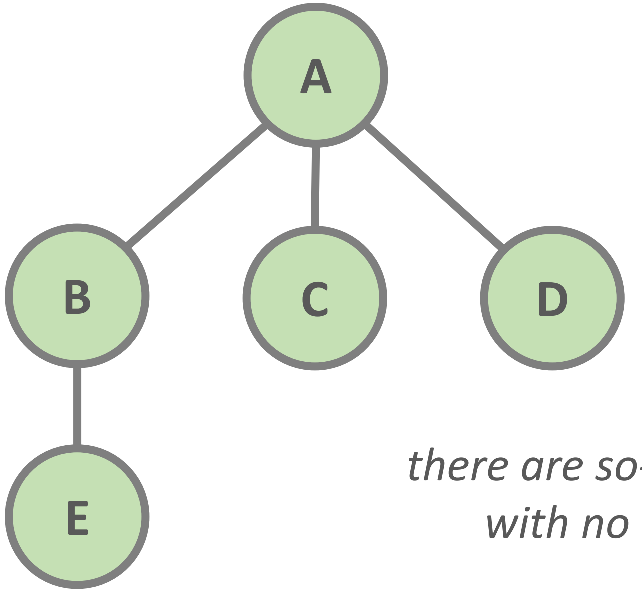
(Algorithms and Data Structures)

Binary Search Trees

- **arrays** can manipulate the last item in **O(1)** constant running time complexity that is quite fast
- **linked lists** can manipulate the first item of the data structure fast
- searching for an arbitrary item takes **O(N)** linear running time for both data structures
- **WHAT IF THE ARRAY DATA STRUCTURE IS SORTED?**
- we can seach for arbitrary item in **O(logN)** logaritmic time complexity
- this is the concept behind **binary search**

Trees (Graph Theory)

*we have access to the **root node** exclusively*
all other nodes can be accessed via the root node

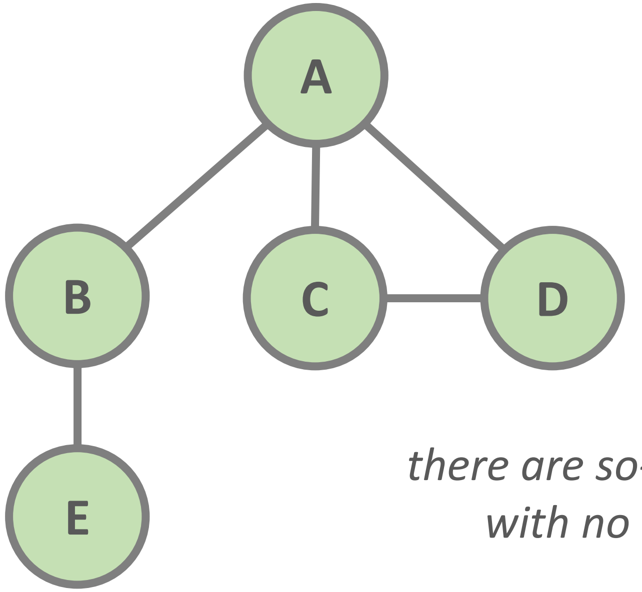


*there are so-called **leaf nodes***
with no children at all

*„A tree is a **G(V,E)** undirected graph in which any two vertices are connected by exactly one path or equivalently a connected acyclic undirected graph"*

Trees (Graph Theory)

*we have access to the **root node** exclusively*
all other nodes can be accessed via the root node



IT IS NOT A TREE !!!

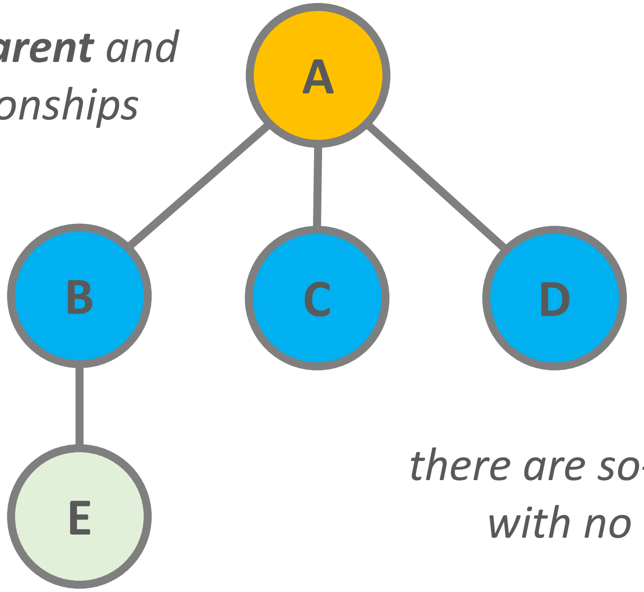
*there are so-called **leaf nodes***
with no children at all

*„A tree is a **G(V,E)** undirected graph in which any two vertices are connected by exactly one path or equivalently a connected acyclic undirected graph"*

Trees (Graph Theory)

*we have access to the **root node** exclusively*
all other nodes can be accessed via the root node

*we can define **parent** and **children** relationships*



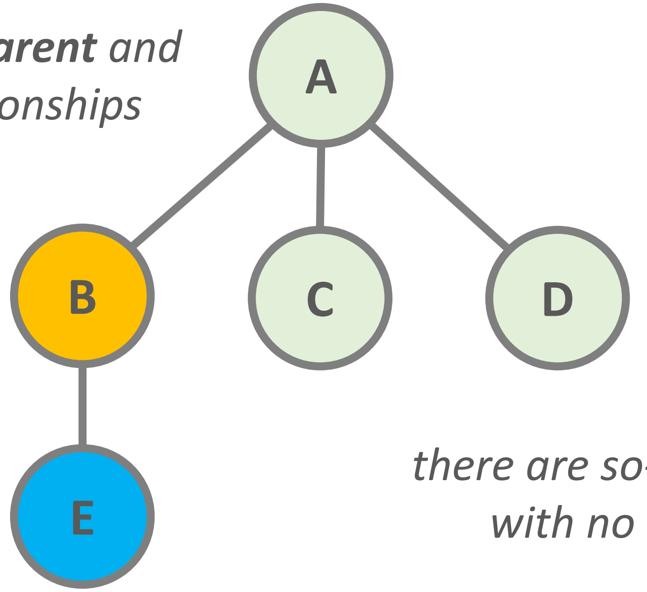
*there are so-called **leaf nodes** with no children at all*

*„A tree is a **G(V,E)** undirected graph in which any two vertices are connected by exactly one path or equivalently a connected acyclic undirected graph”*

Trees (Graph Theory)

we have access to the ***root node*** *exclusively*
all other nodes can be accessed via the root node

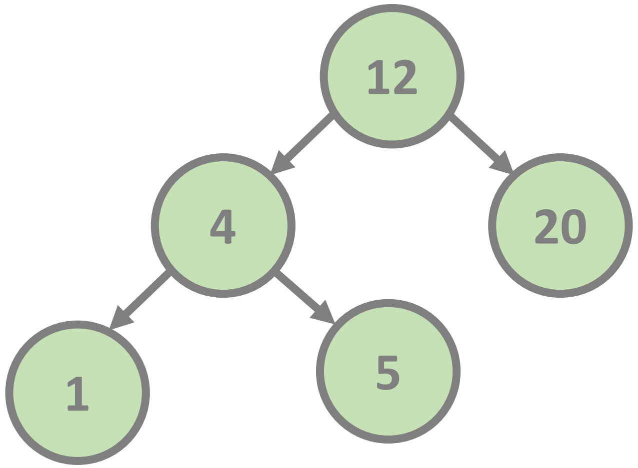
we can define ***parent*** *and*
children *relationships*



there are so-called ***leaf nodes***
with no children at all

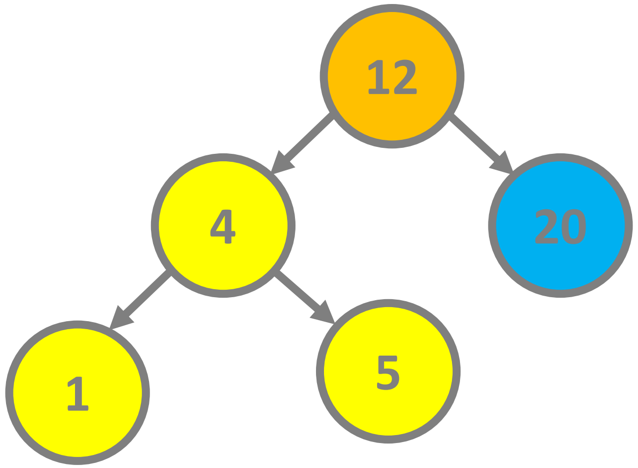
„A tree is a ***G(V,E)*** *undirected graph in which any two vertices are connected by exactly one path or equivalently a connected acyclic undirected graph"*

Binary Search Trees



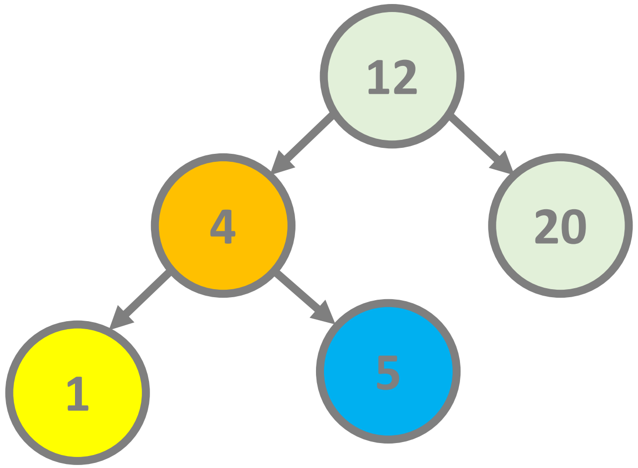
- every node in the tree can have at most **2** children (**left child** and **right child**)
- **left child** is smaller than the parent node
- **right child** is greater than the parent node
- we can access the **root node** exclusively and all other nodes can be accessed via the root node

Binary Search Trees



- every node in the tree can have at most **2** children (**left child** and **right child**)
- **left child** is smaller than the parent node
- **right child** is greater than the parent node
- we can access the **root node** exclusively and all other nodes can be accessed via the root node

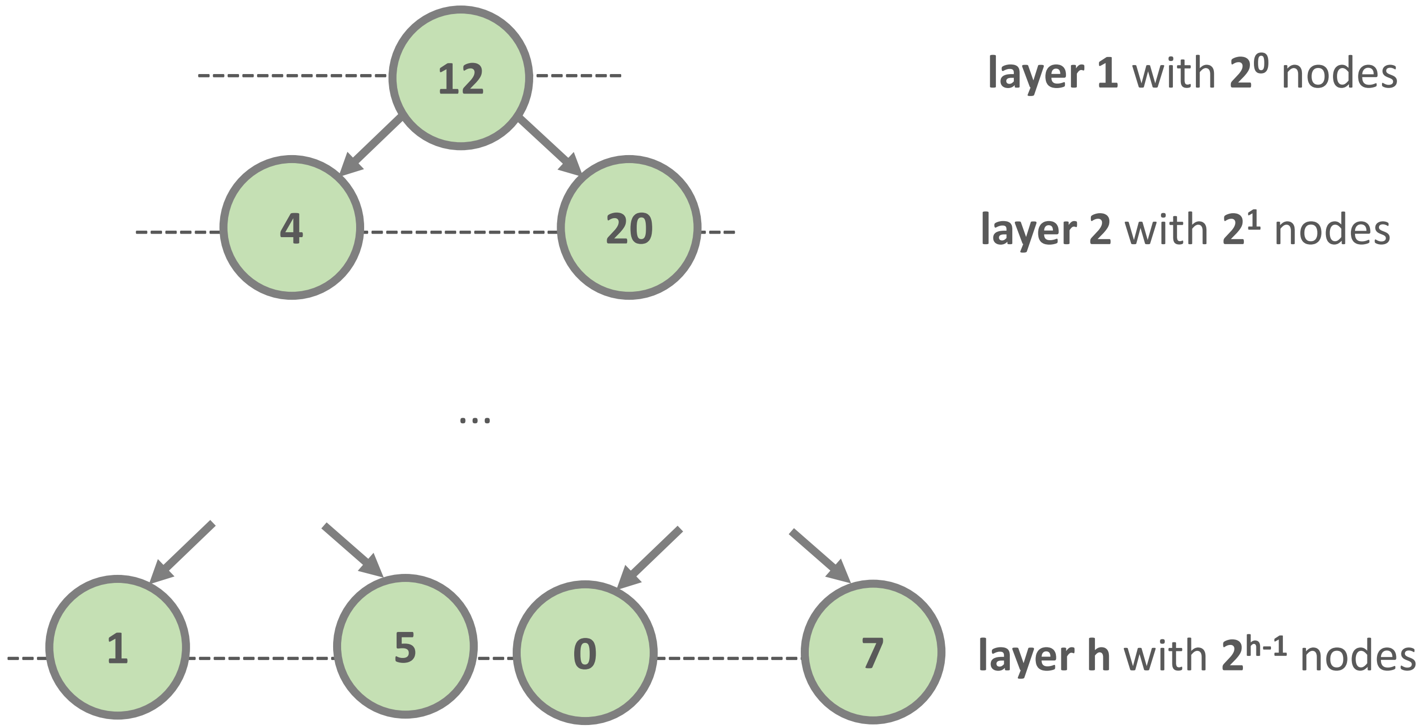
Binary Search Trees



- every node in the tree can have at most **2** children (**left child** and **right child**)
- **left child** is smaller than the parent node
- **right child** is greater than the parent node
- we can access the **root node** exclusively and all other nodes can be accessed via the root node

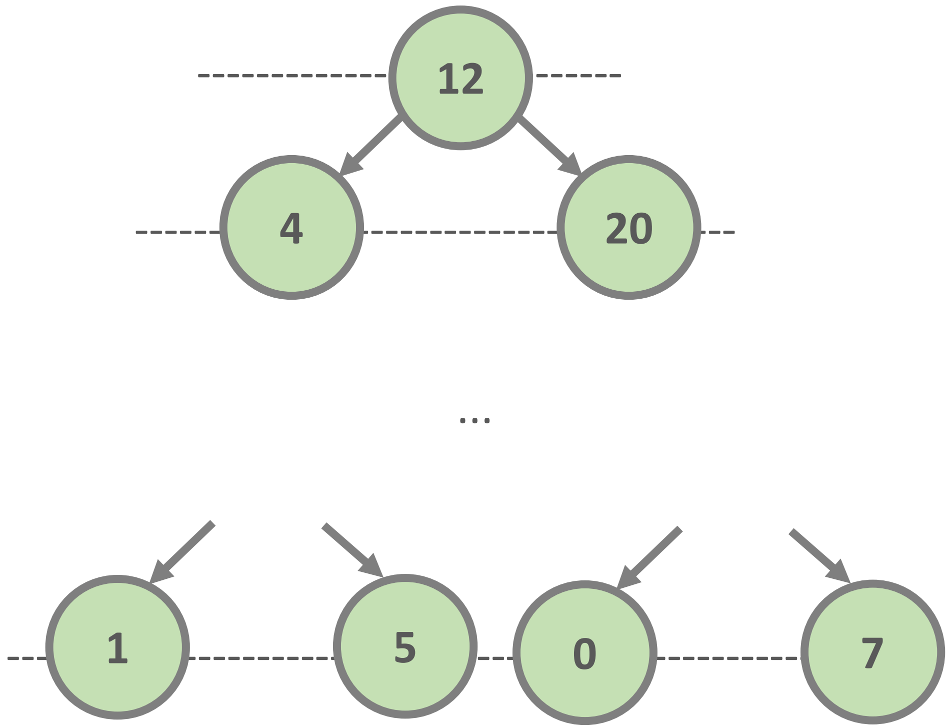
EVERY DECISION CAN GET RID OF HALF OF THE DATA (LIKE WITH BINARY SEARCH) AND THIS IS HOW WE CAN ACHIEVE O(logN) RUNNING TIME

Binary Search Trees



The **height of a tree** is the number of edges on the longest downward path between the **root** and a **leaf node**. The number of layers the tree contains.

Binary Search Trees



layer 1 with $\mathbf{2^0}$ nodes

layer 2 with $\mathbf{2^1}$ nodes

layer h with $\mathbf{2^{h-1}}$ nodes

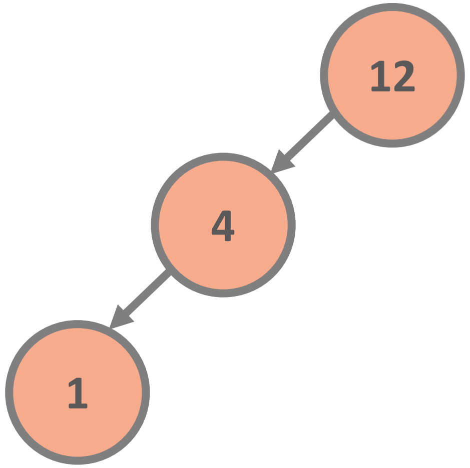
*how many **N** nodes are there in a complete binary search tree with **h** height?*

$$2^{h-1} = N$$
$$\log_2 2^{h-1} = \log_2 N$$
$$h = \log_2 N + 1$$
$$h = O(\log N)$$

Binary Search Trees

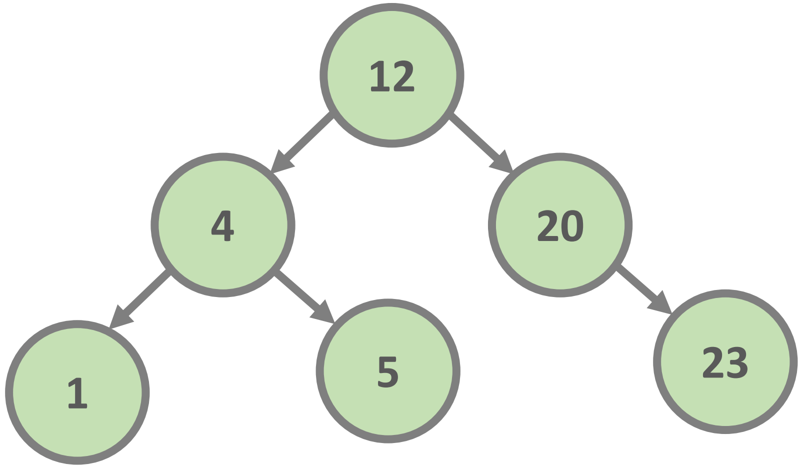
- the logarithmic **O(logN)** running time is valid only when the tree structure is balanced
- we should keep the height of a tree at a minimum which is **h=logN**
- the tree structure may became **imbalanced** which means the number of nodes significantly differ in the subtrees
- if the tree is imbalanced so the **h=logN** relation is no more valid then the operations' running time is no more **O(logN)** logarithmic

Binary Search Trees



IMBALANCED TREE

*in an **imbalanced tree** the running time of operations can be reduced to even **O(N)** linear Running time complexity*



BALANCED TREE

*in a **balanced tree** the running time of operations are **O(logN)** always*

Binary Search Trees

- **binary search trees** are data structures so the aim is to be able to store items efficiently
- it keeps the keys in sorted order so that lookup and other operations can use the principle of binary search with **O(logN)** running time
- each comparison allows the operations to skip over half of the tree, so that each operation takes time **proportional to the logarithm** of the number of items stored in the tree
- this is much better than **O(N)** the linear time required to find items by key in an unsorted array but slower than the corresponding operations on hash tables with **O(1)**

Binary Search Trees

(Algorithms and Data Structures)

Binary Search Trees

INSERT(12)

Binary Search Trees

INSERT(12)

Binary Search Trees

Binary Search Trees

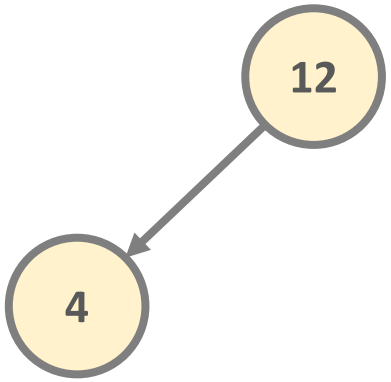
INSERT(4)

Binary Search Trees

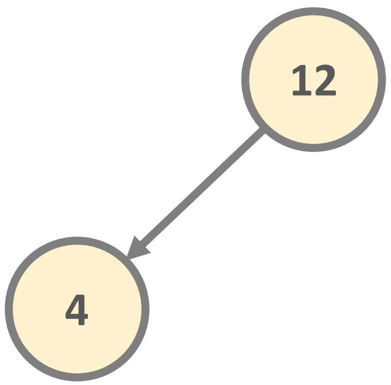
INSERT(4)

Binary Search Trees

INSERT(4)

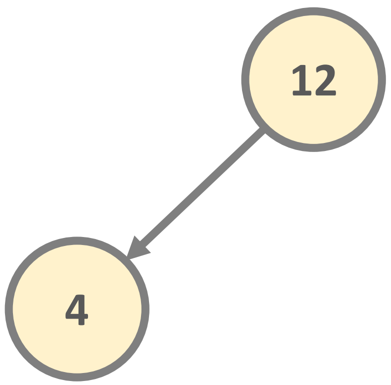


Binary Search Trees



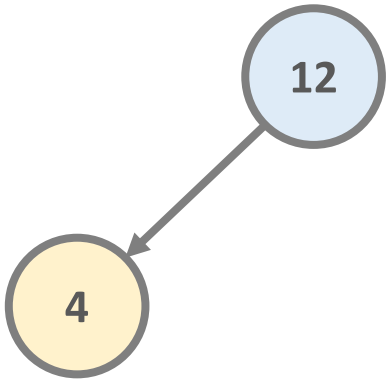
Binary Search Trees

INSERT(8)



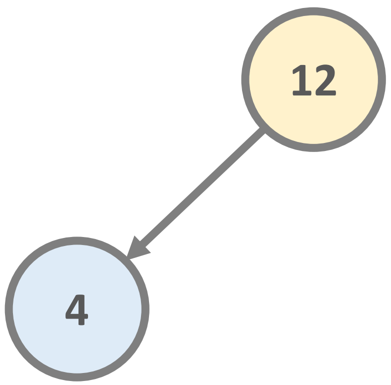
Binary Search Trees

INSERT(8)



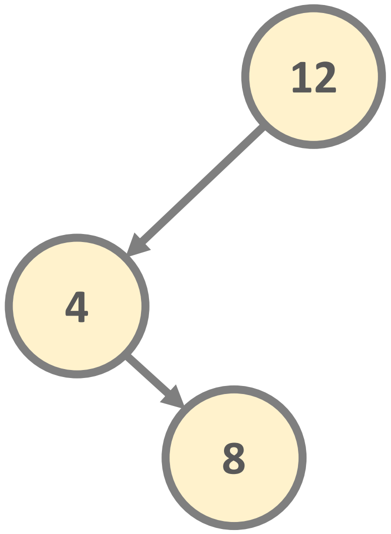
Binary Search Trees

INSERT(8)

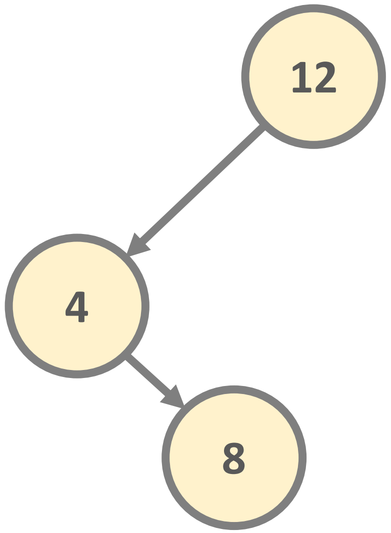


Binary Search Trees

INSERT(8)

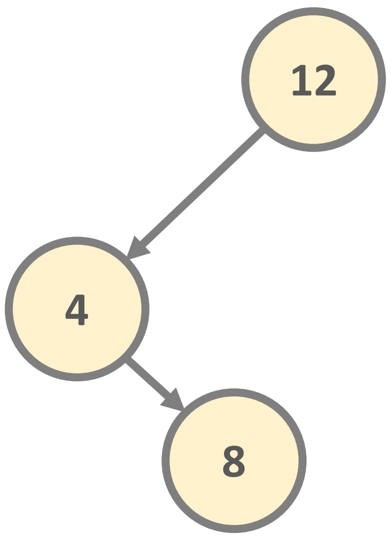


Binary Search Trees



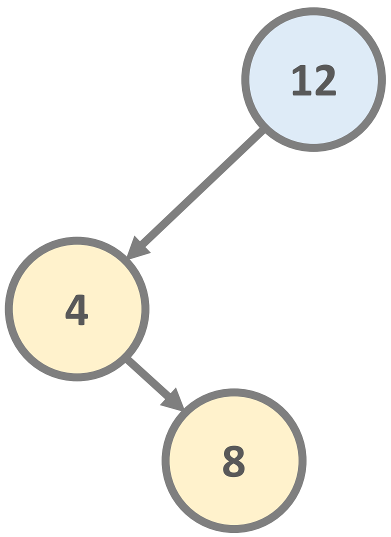
Binary Search Trees

INSERT(20)



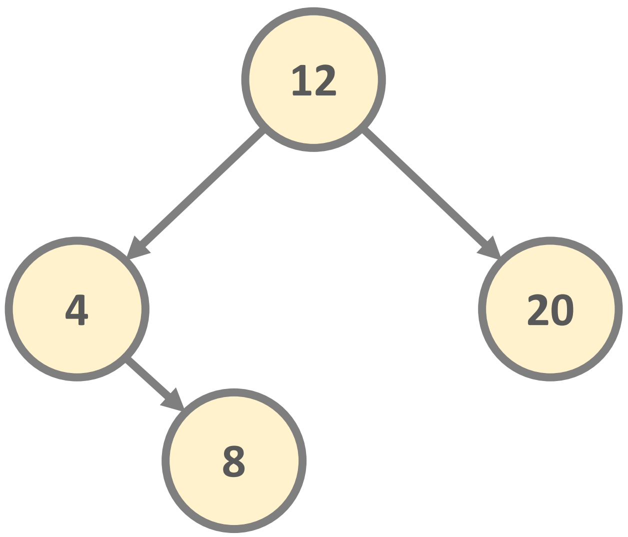
Binary Search Trees

INSERT(20)

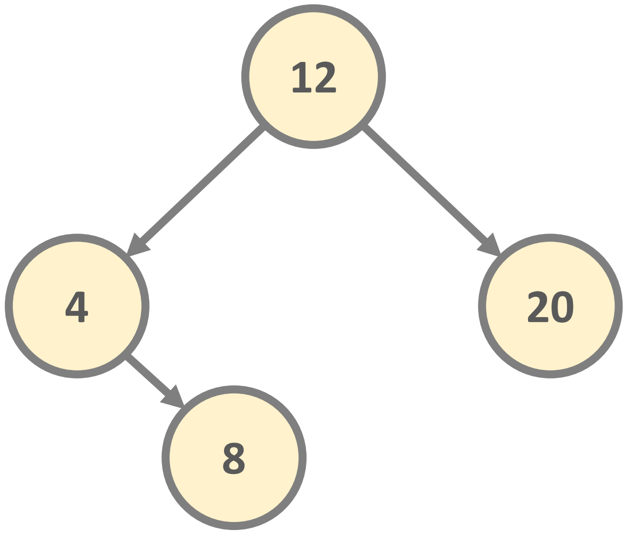


Binary Search Trees

INSERT(20)

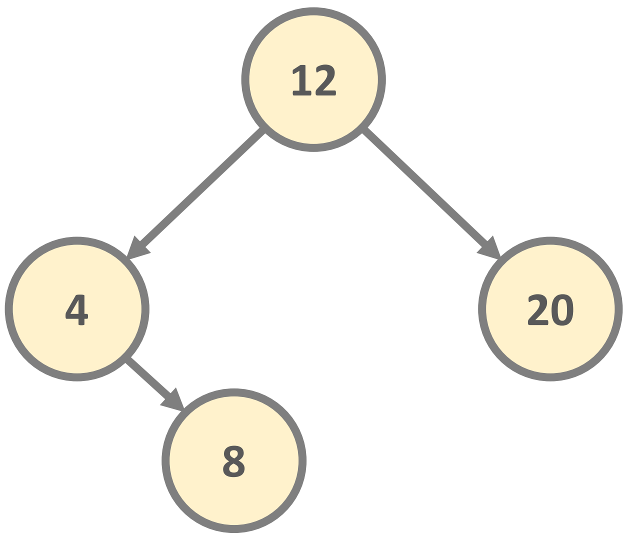


Binary Search Trees



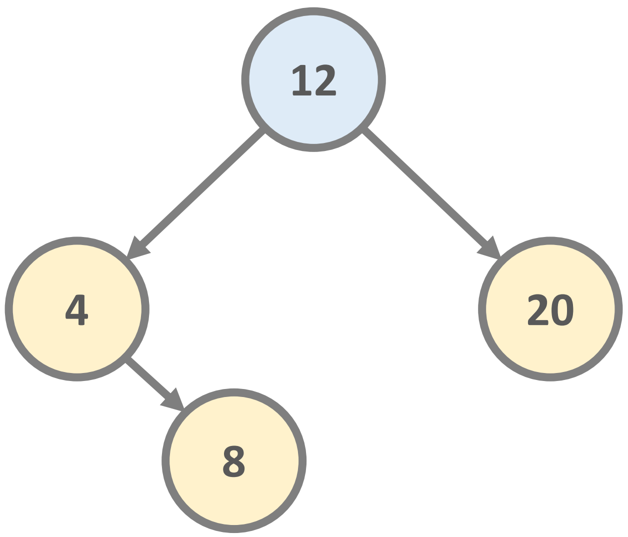
Binary Search Trees

INSERT(27)



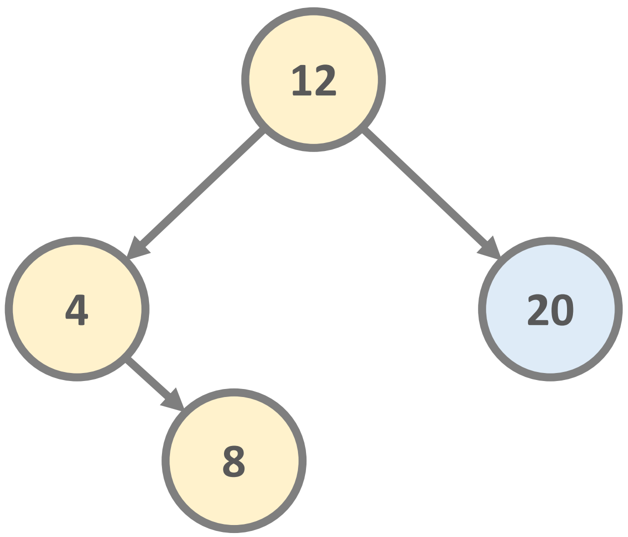
Binary Search Trees

INSERT(27)



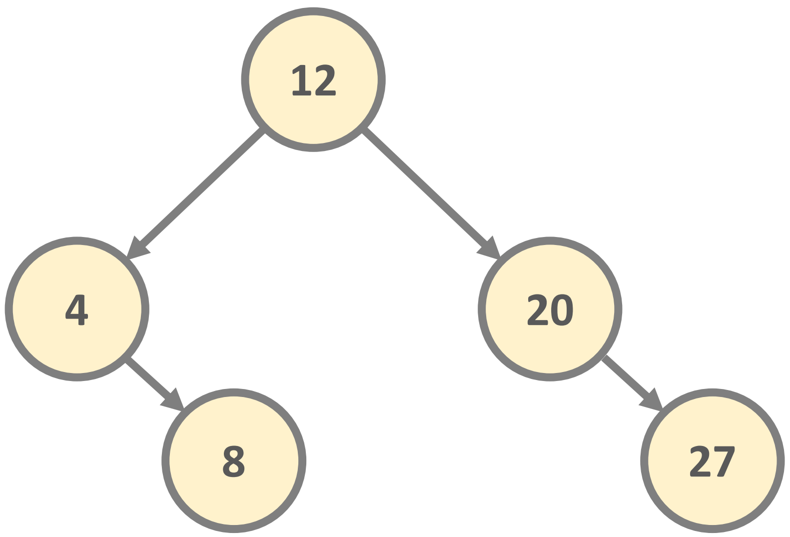
Binary Search Trees

INSERT(27)

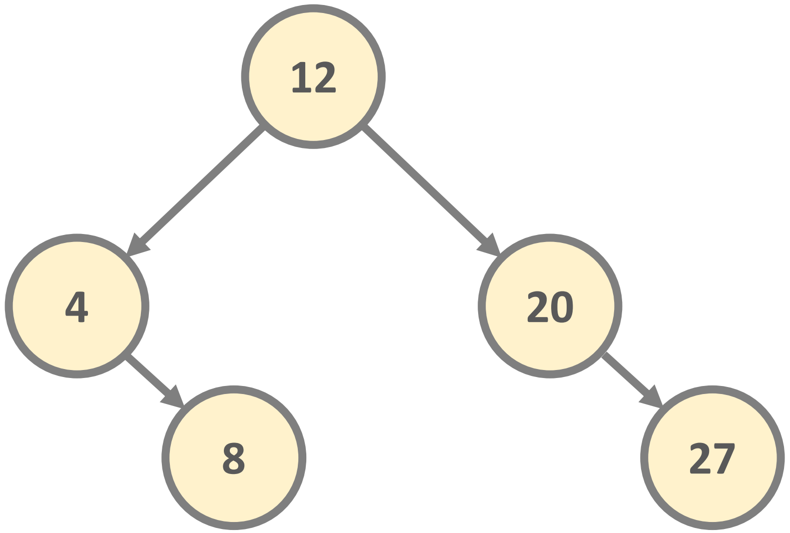


Binary Search Trees

INSERT(27)

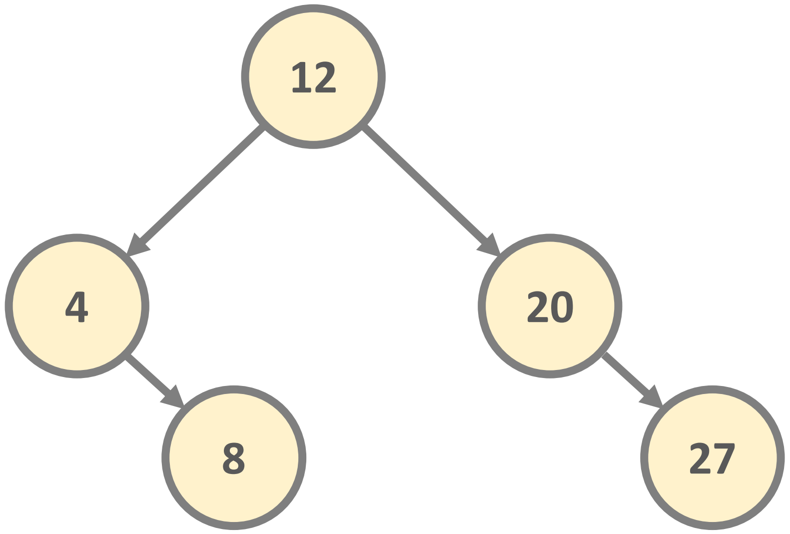


Binary Search Trees



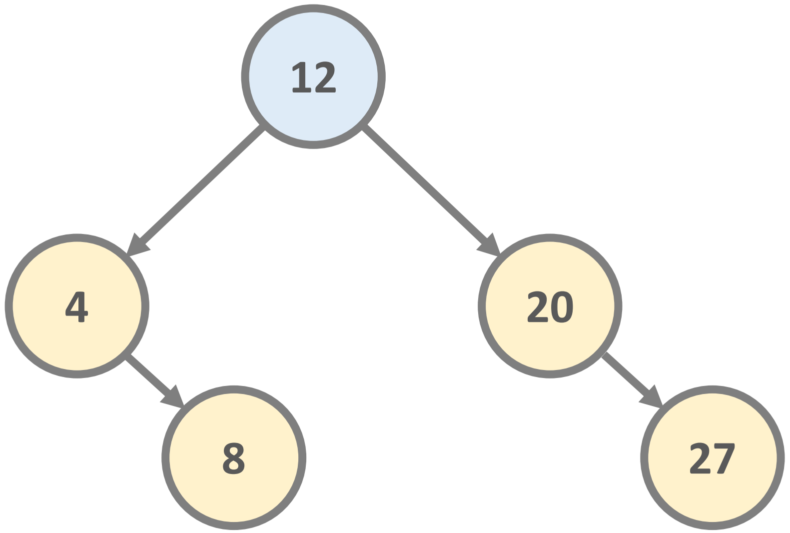
Binary Search Trees

INSERT(16)



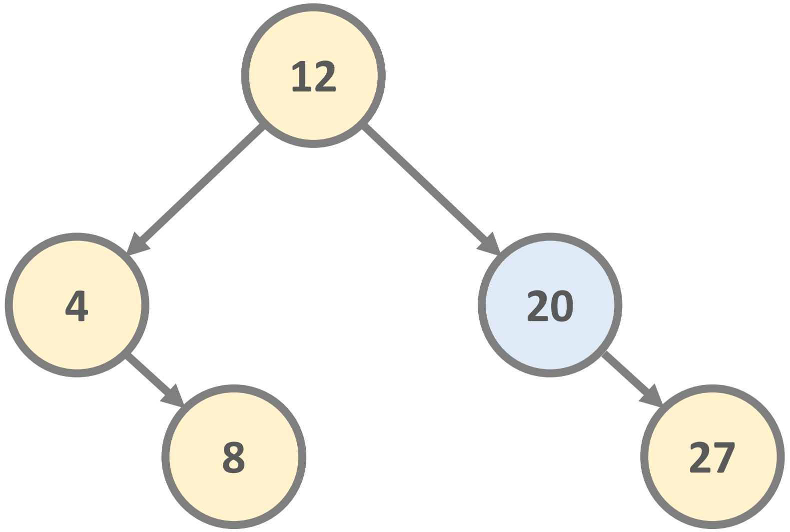
Binary Search Trees

INSERT(16)



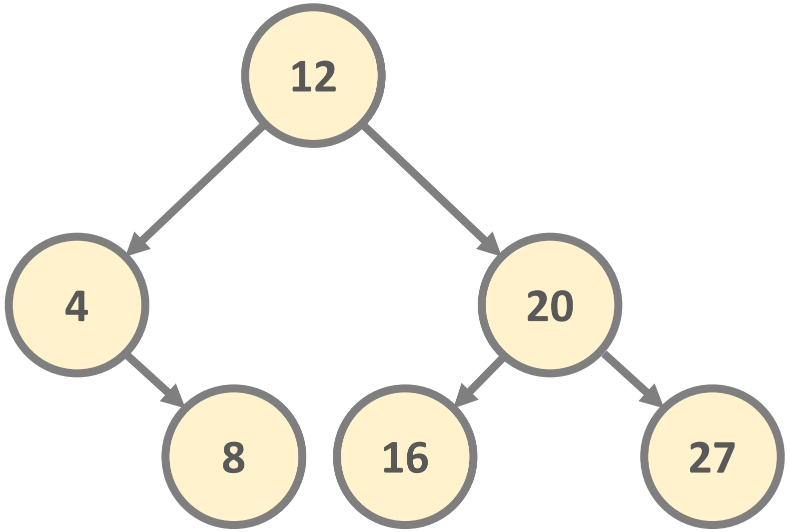
Binary Search Trees

INSERT(16)

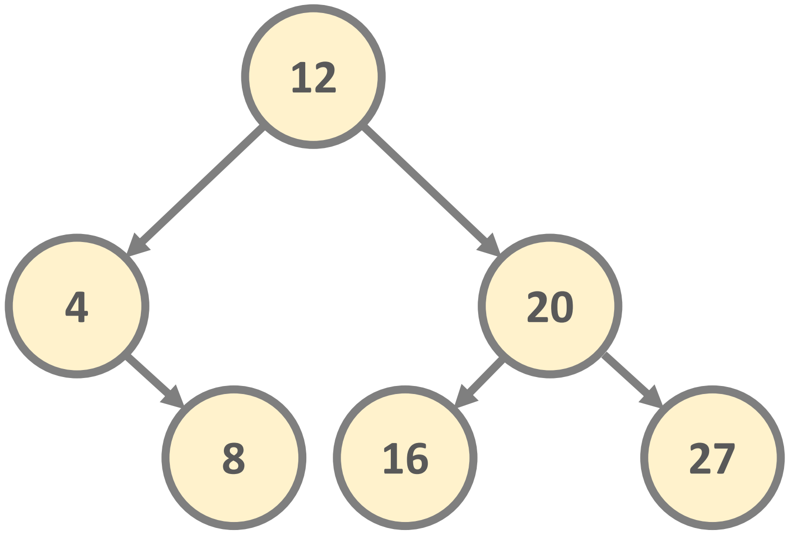


Binary Search Trees

INSERT(16)

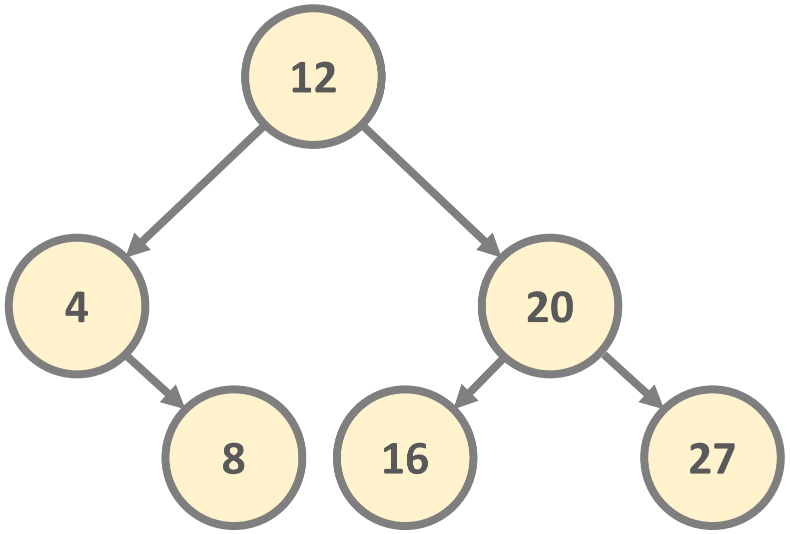


Binary Search Trees



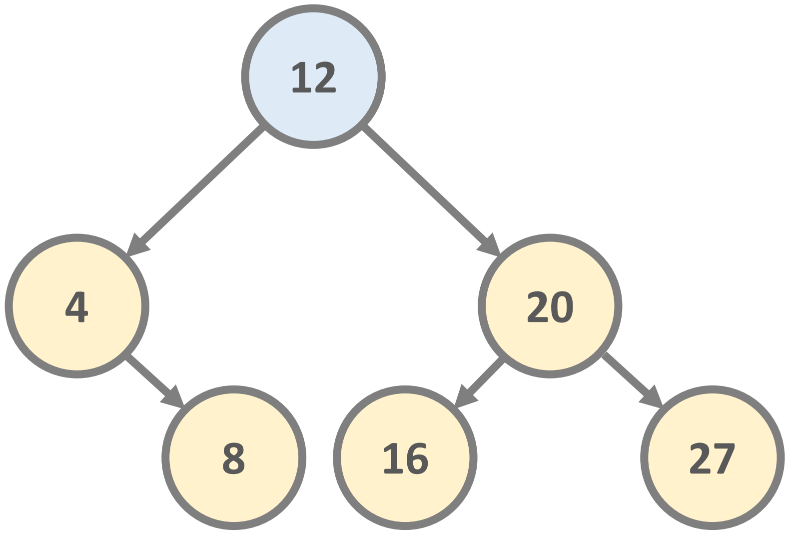
Binary Search Trees

INSERT(1)



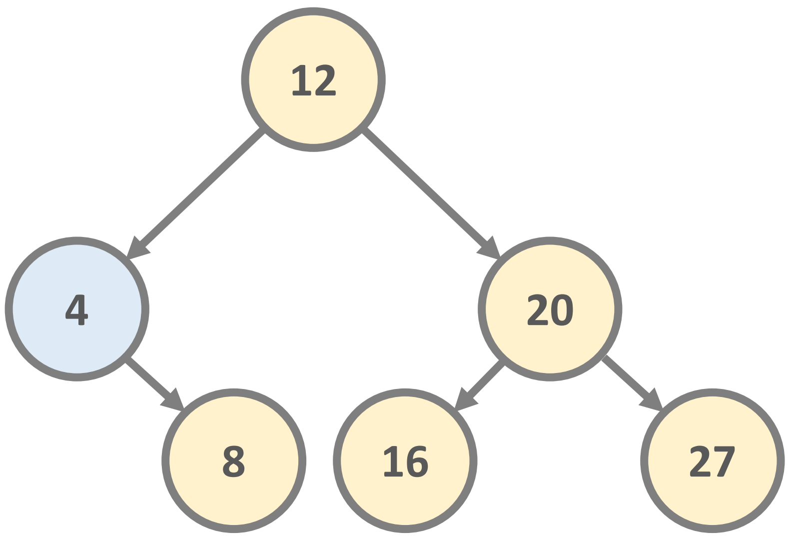
Binary Search Trees

INSERT(1)



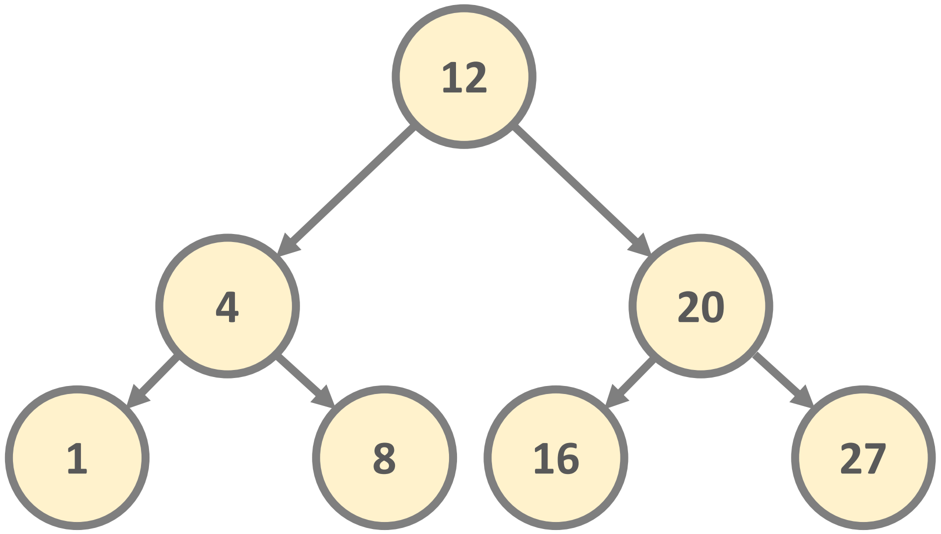
Binary Search Trees

INSERT(1)

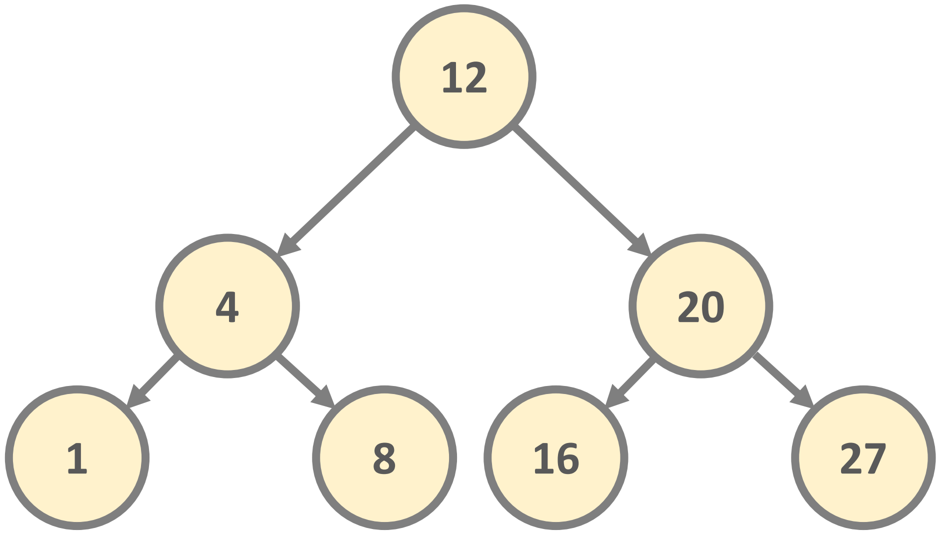


Binary Search Trees

INSERT(1)

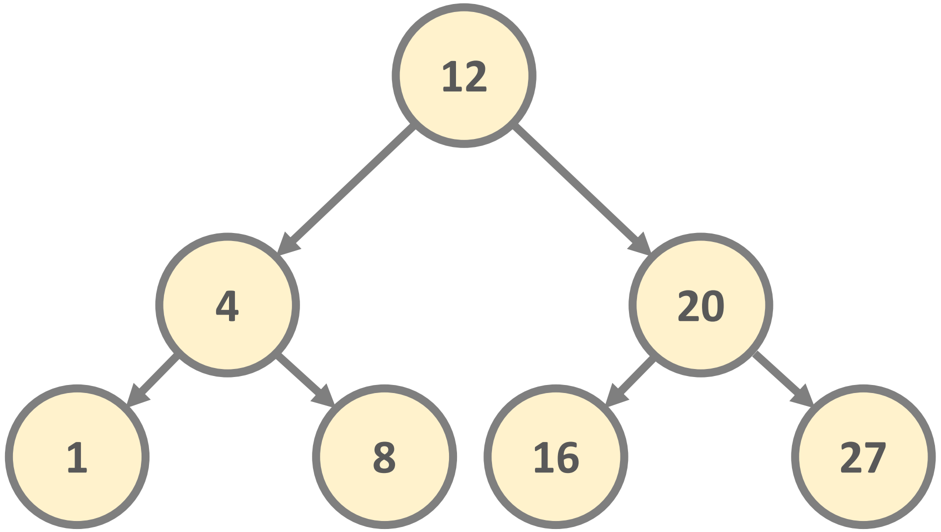


Binary Search Trees



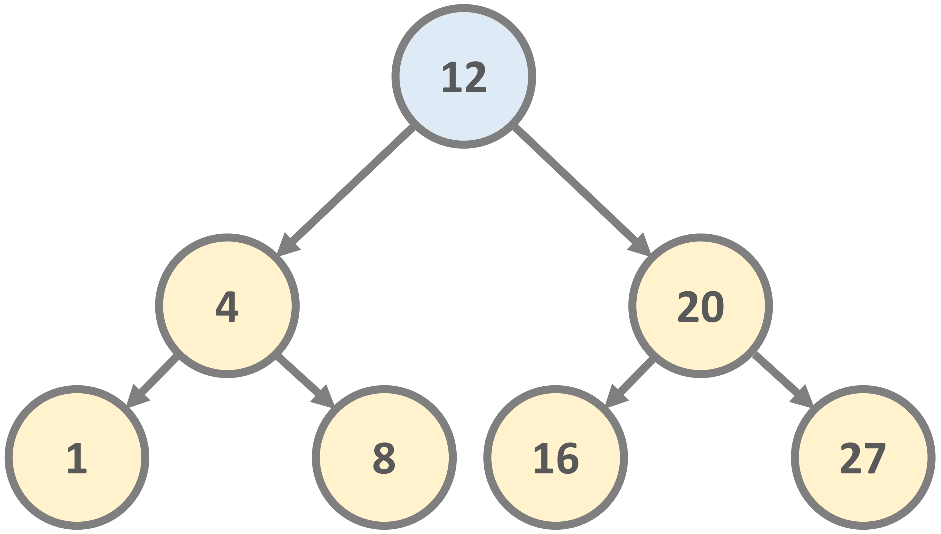
Binary Search Trees

SEARCH(8)



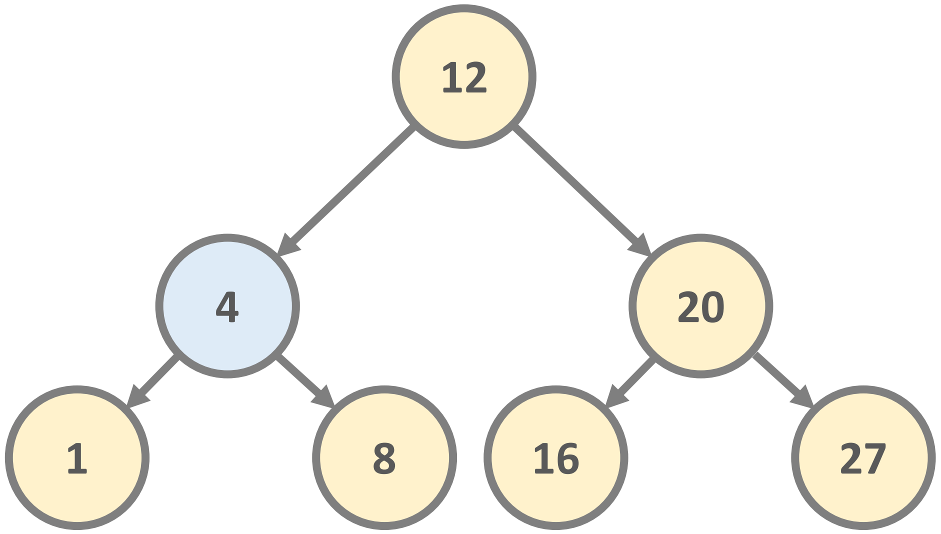
Binary Search Trees

SEARCH(8)



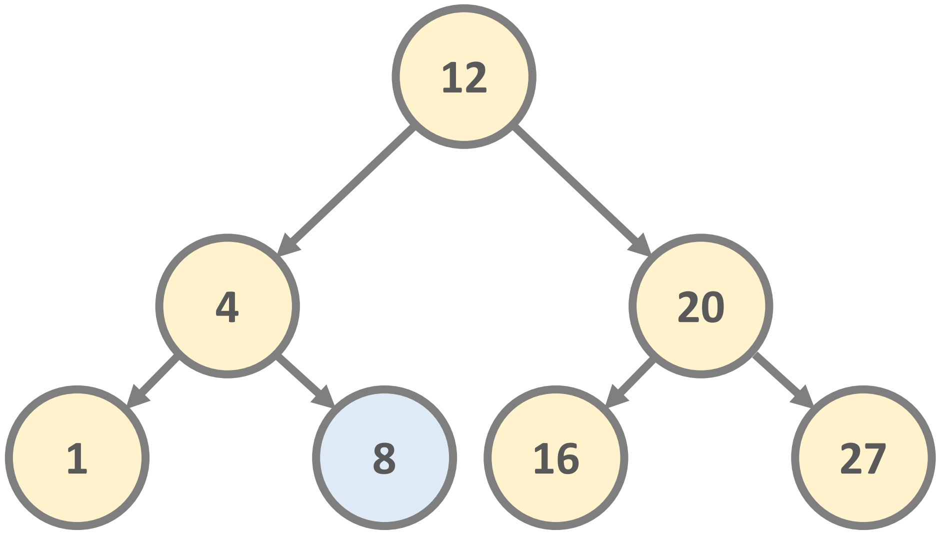
Binary Search Trees

SEARCH(8)



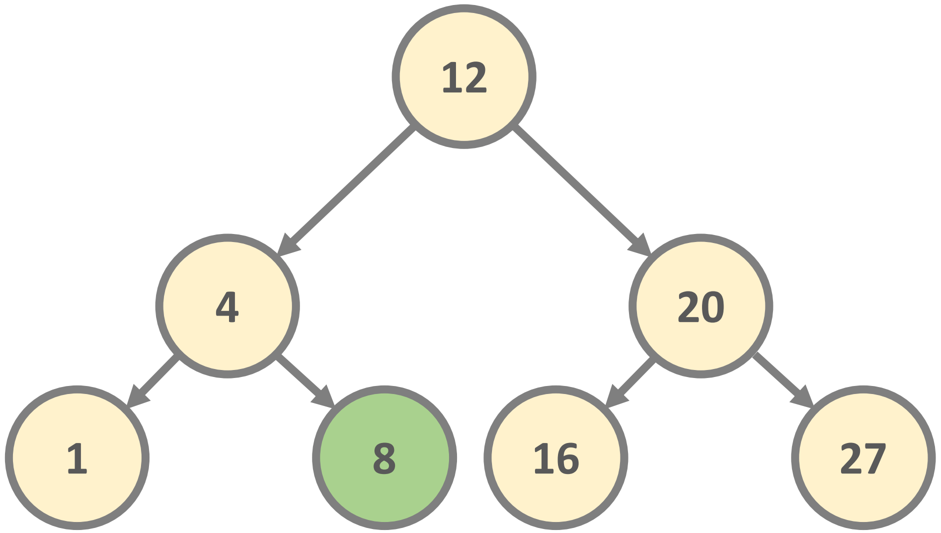
Binary Search Trees

SEARCH(8)



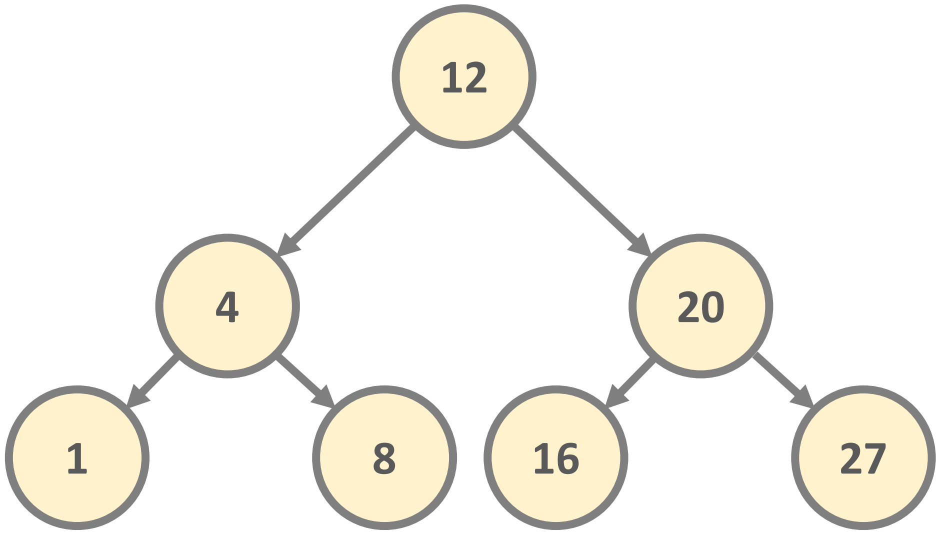
Binary Search Trees

SEARCH(8)



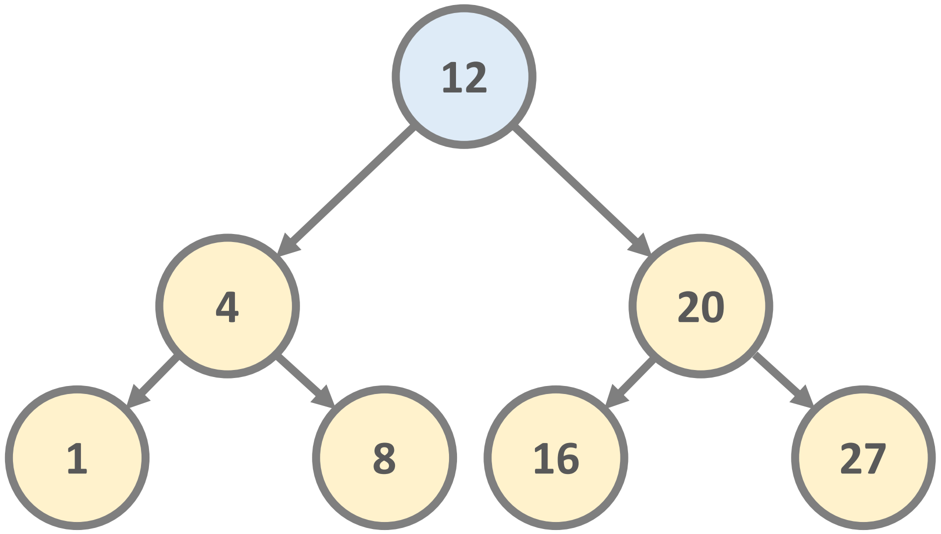
Binary Search Trees

SEARCH MAX()



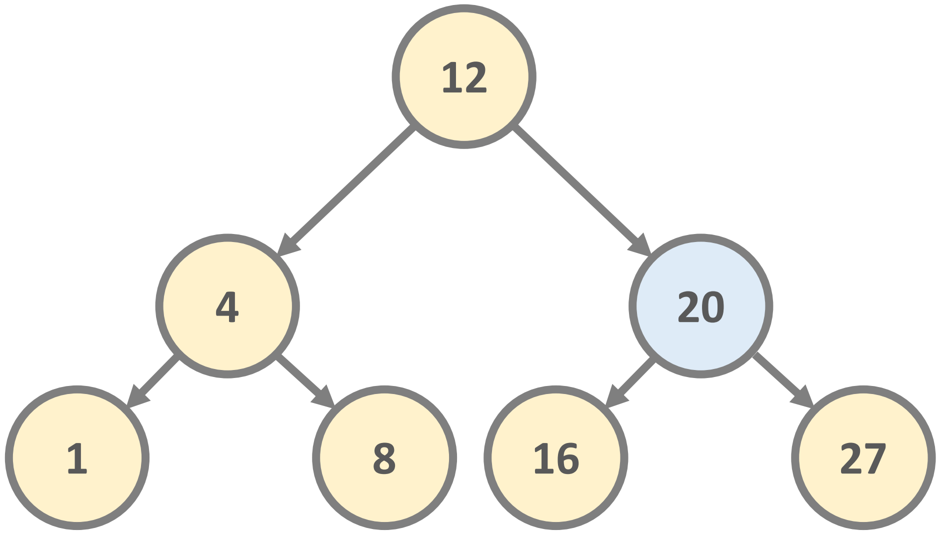
Binary Search Trees

SEARCH MAX()



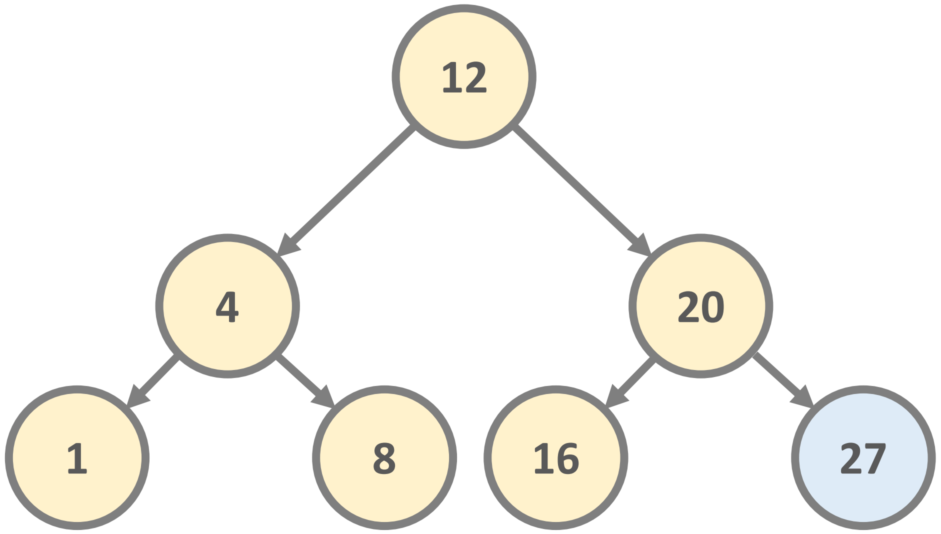
Binary Search Trees

SEARCH MAX()



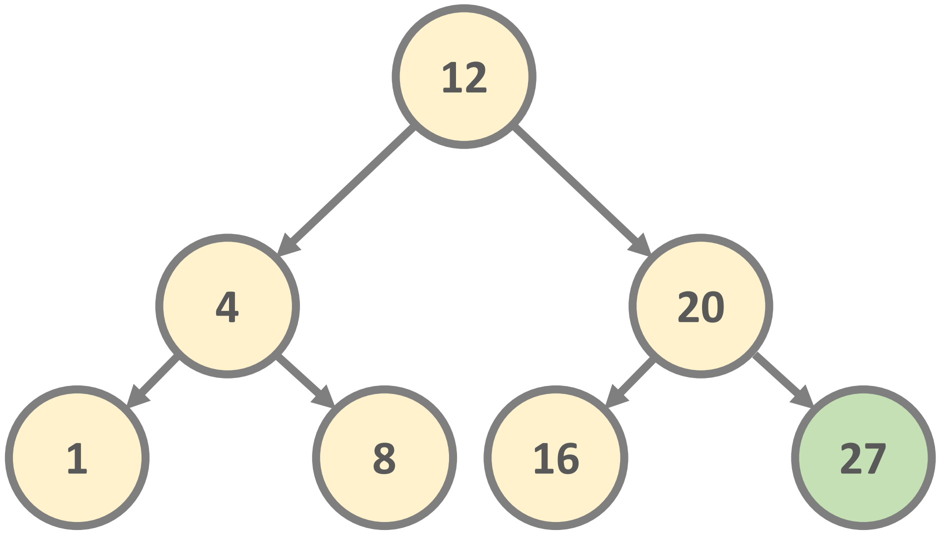
Binary Search Trees

SEARCH MAX()



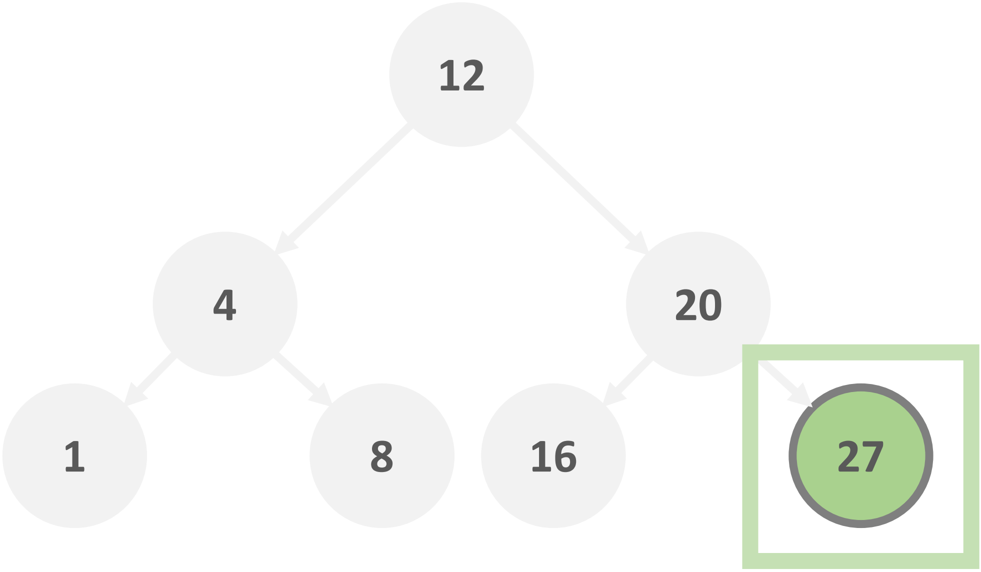
Binary Search Trees

SEARCH MAX()



Binary Search Trees

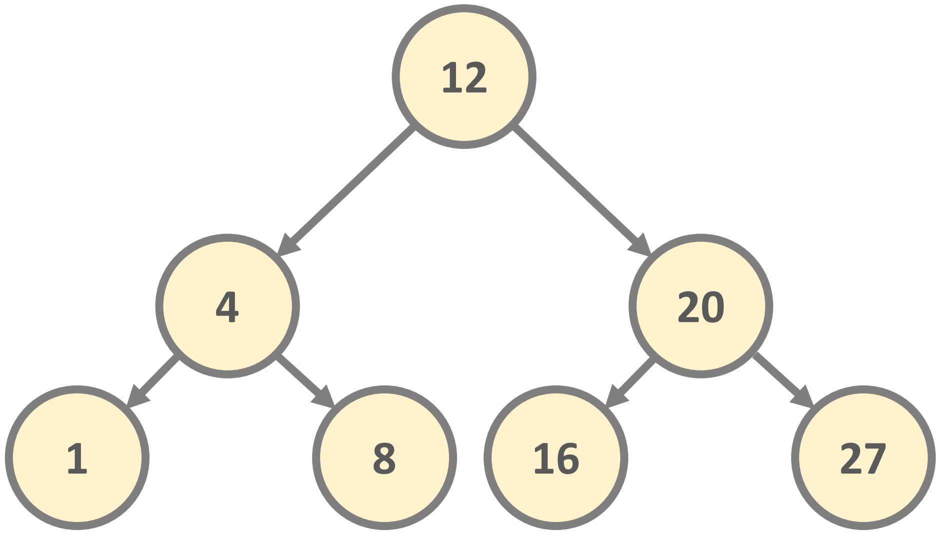
SEARCH MAX()



*the **maximum** item in the binary search tree*
*is the **rightmost** item in the tree*

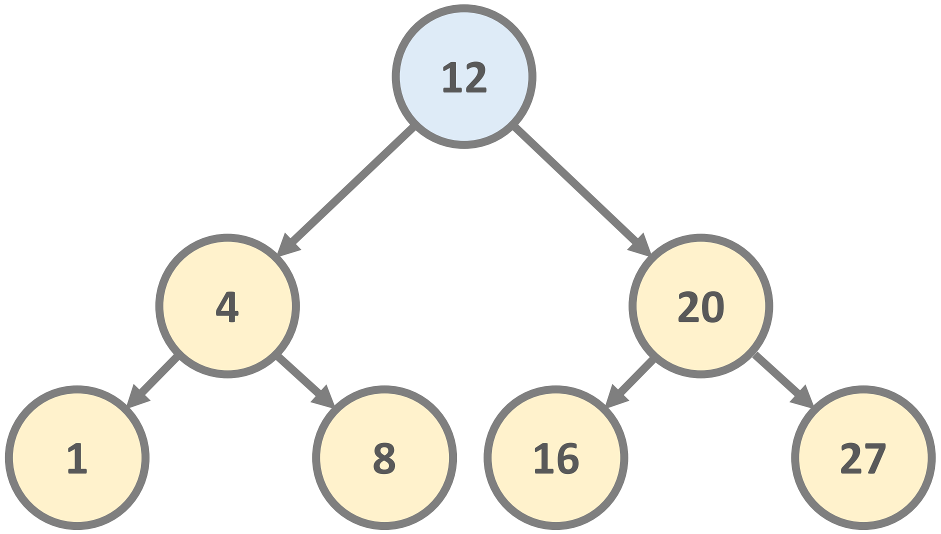
Binary Search Trees

SEARCH MIN()



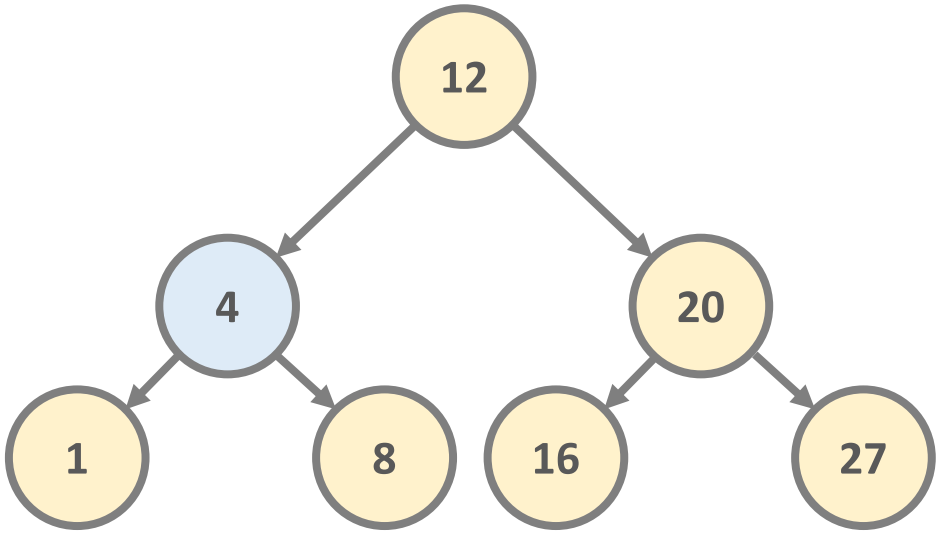
Binary Search Trees

SEARCH MIN()



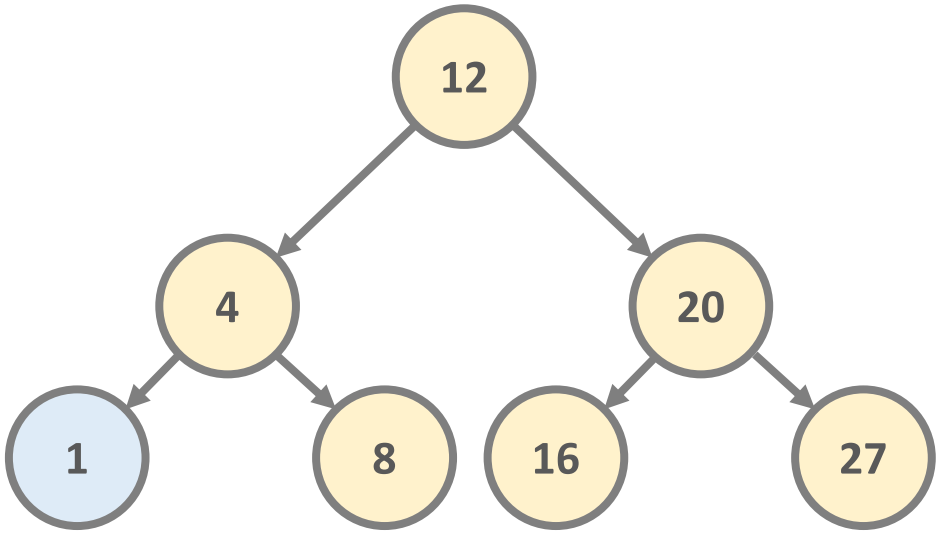
Binary Search Trees

SEARCH MIN()



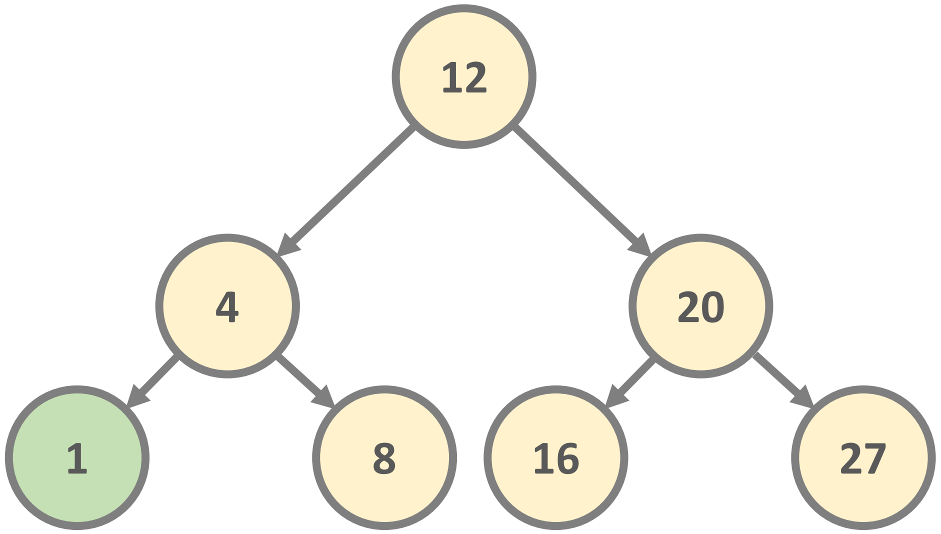
Binary Search Trees

SEARCH MIN()



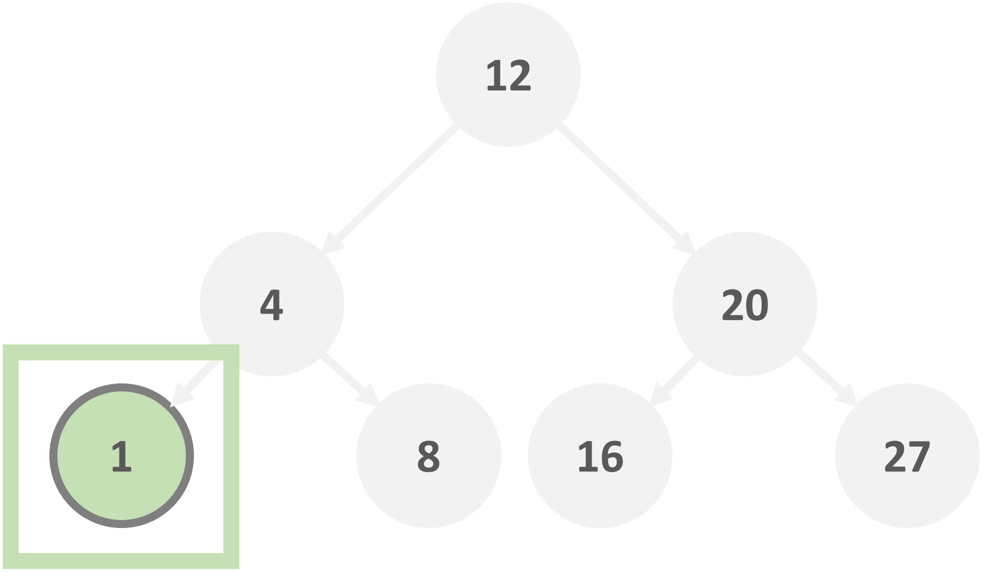
Binary Search Trees

SEARCH MIN()



Binary Search Trees

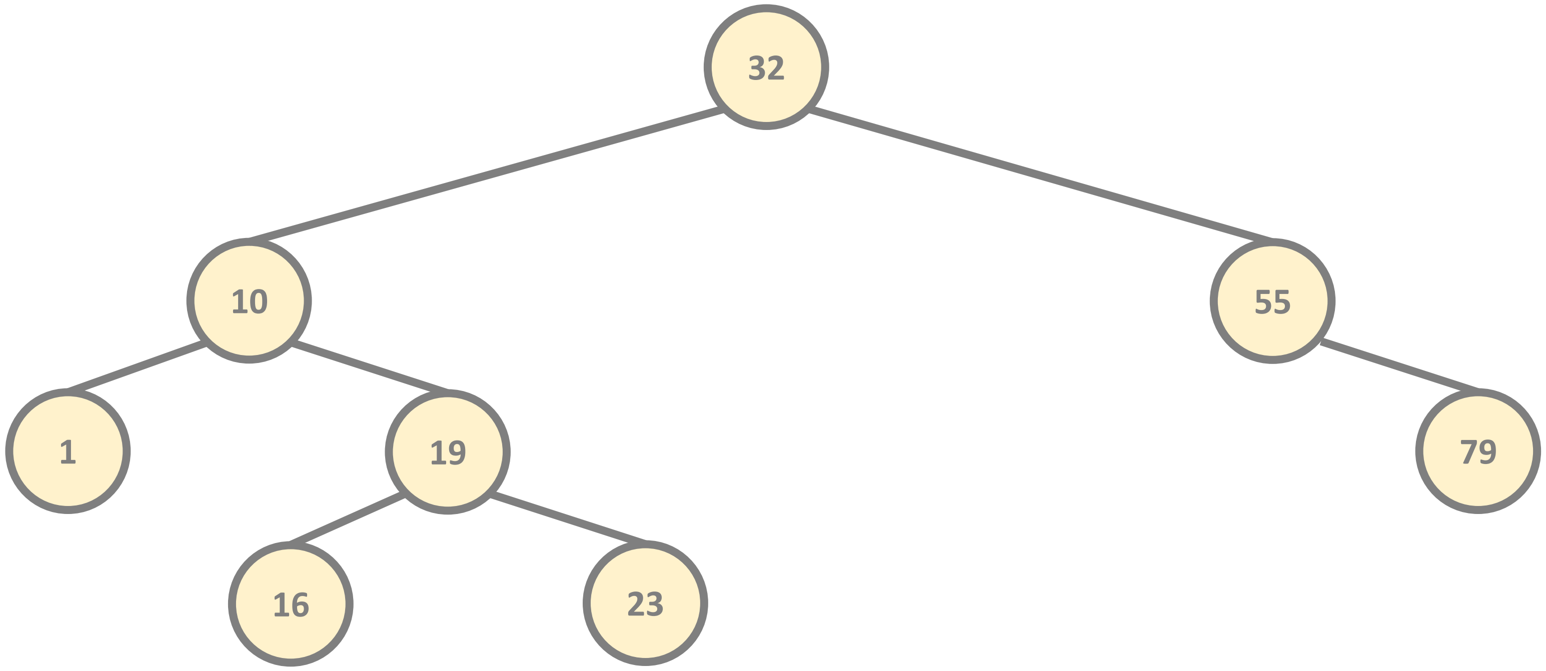
SEARCH MIN()



*the **minimum** item in the binary search tree*
*is the **leftmost** item in the tree*

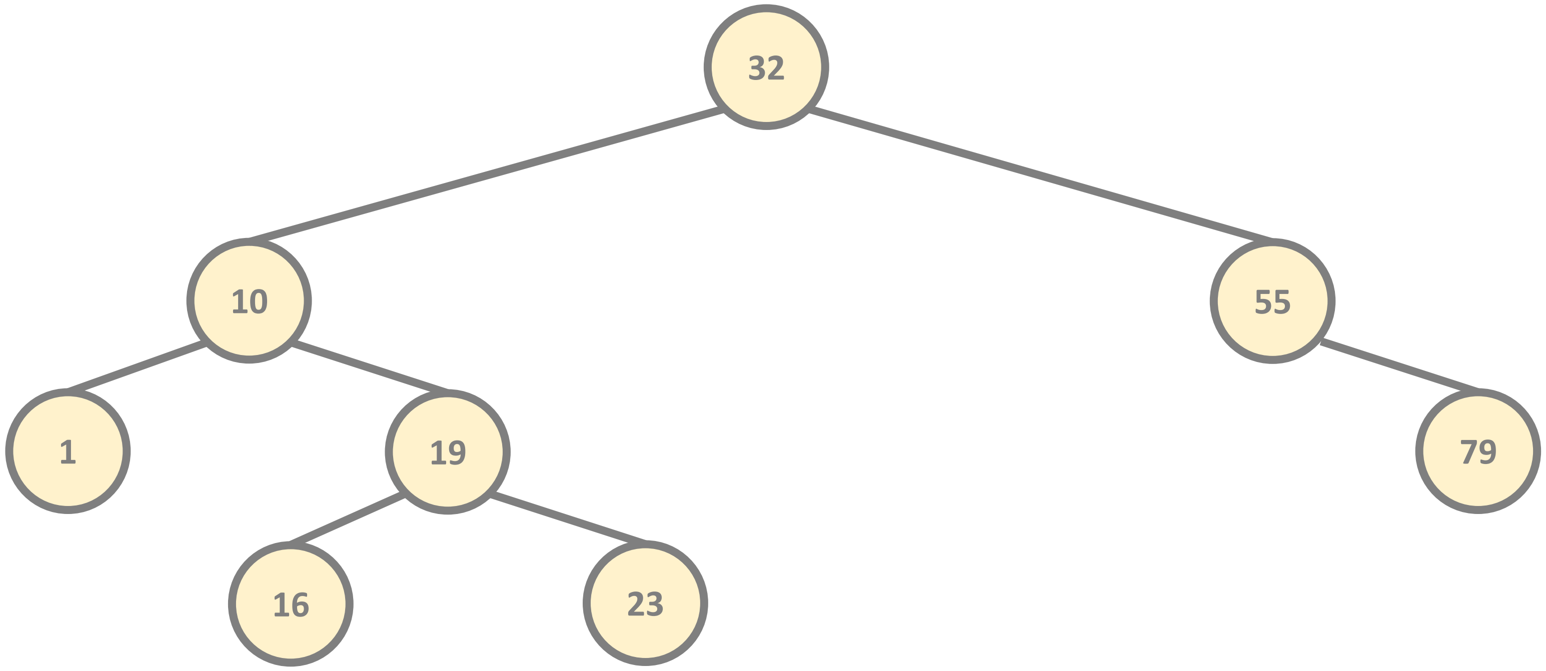
Binary Search Trees
(Algorithms and Data Structures)

Binary Search Trees



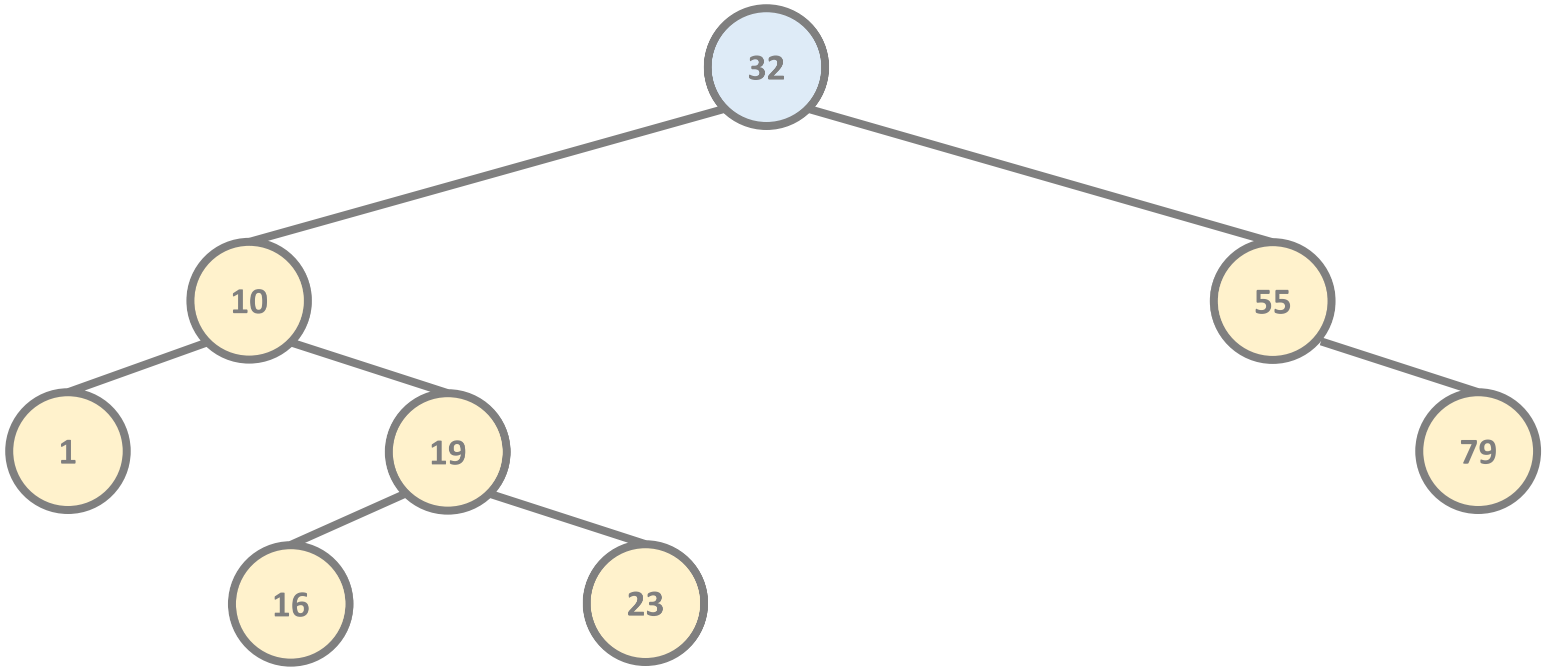
Binary Search Trees

1.) REMOVING A LEAF NODE



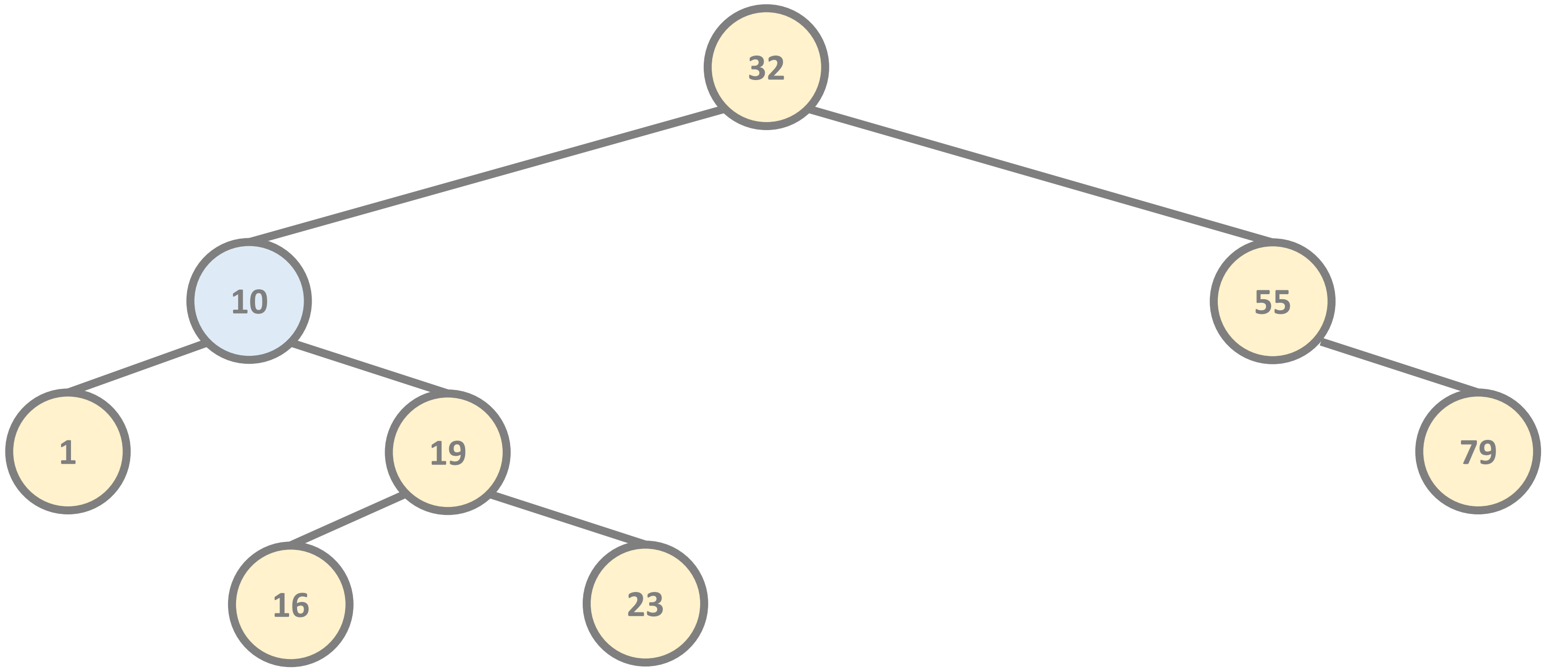
Binary Search Trees

1.) REMOVING A LEAF NODE



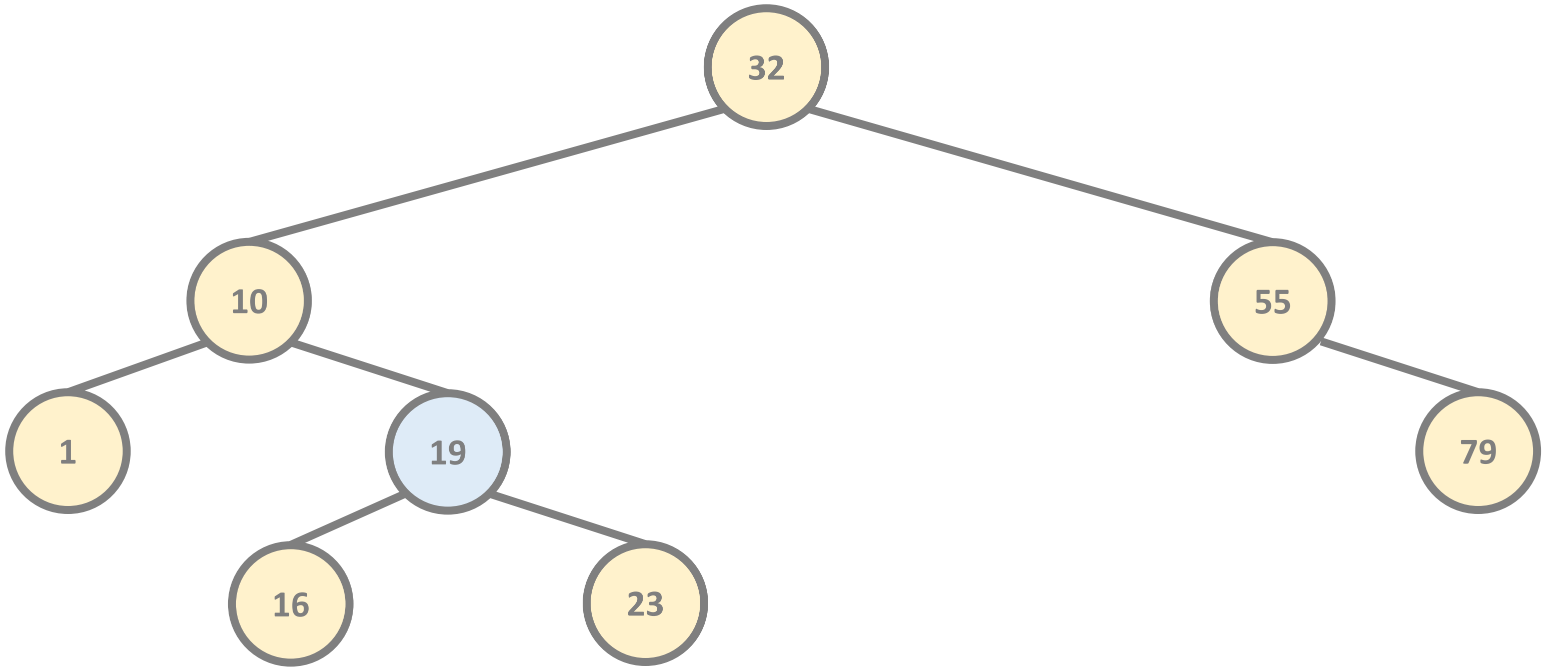
Binary Search Trees

1.) REMOVING A LEAF NODE



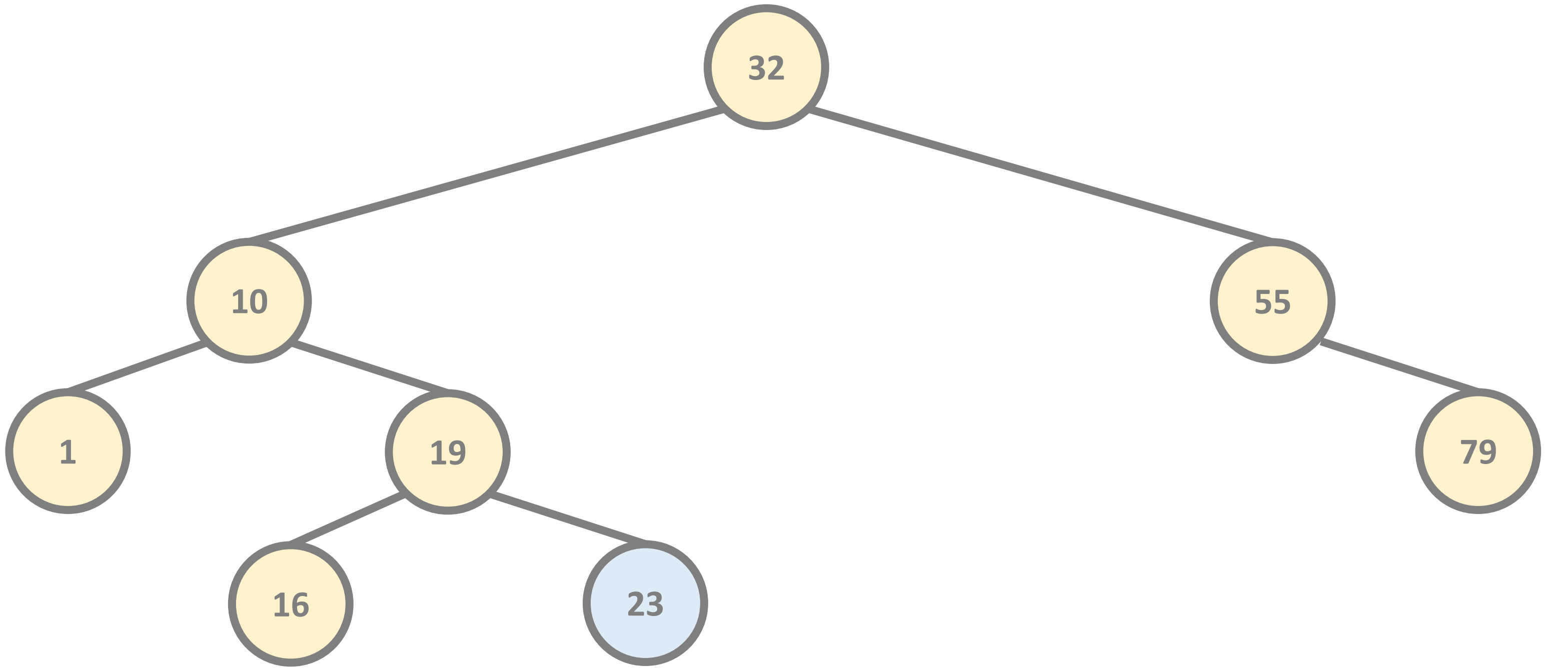
Binary Search Trees

1.) REMOVING A LEAF NODE



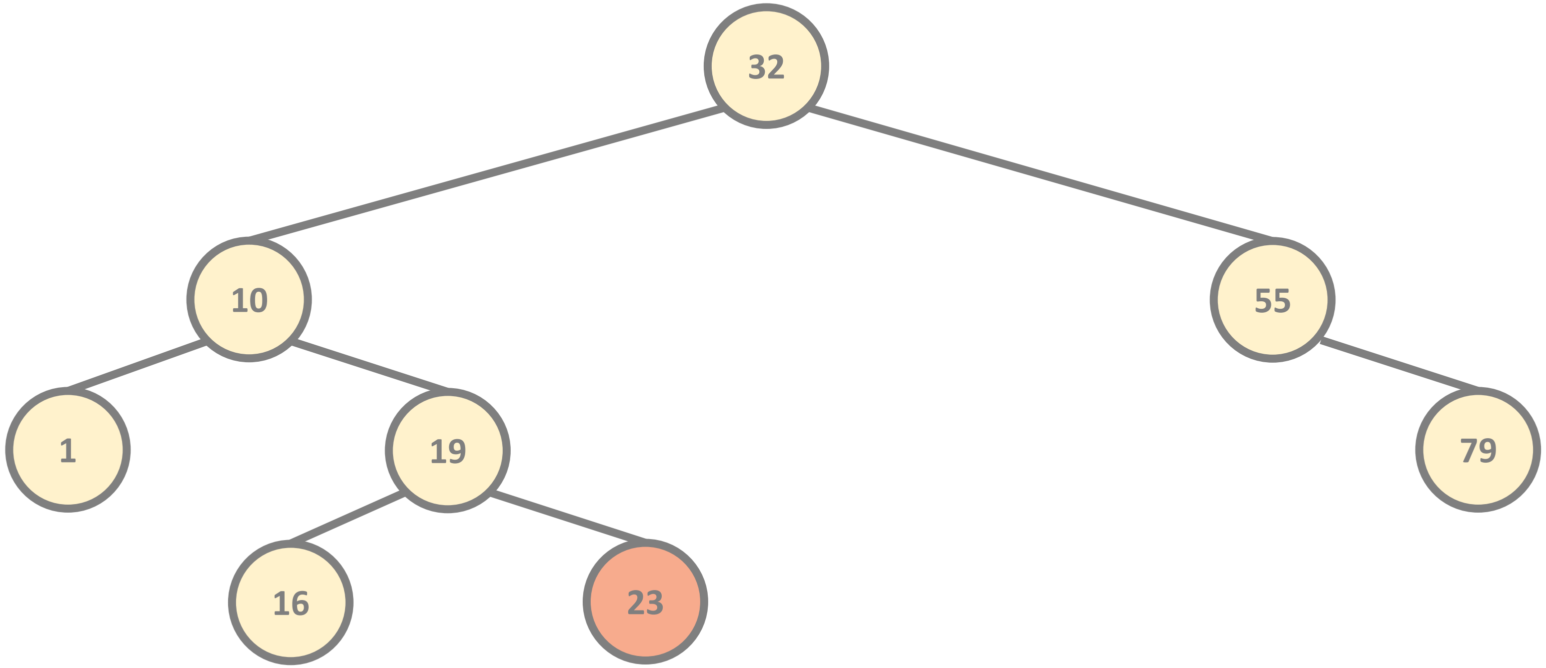
Binary Search Trees

1.) REMOVING A LEAF NODE



Binary Search Trees

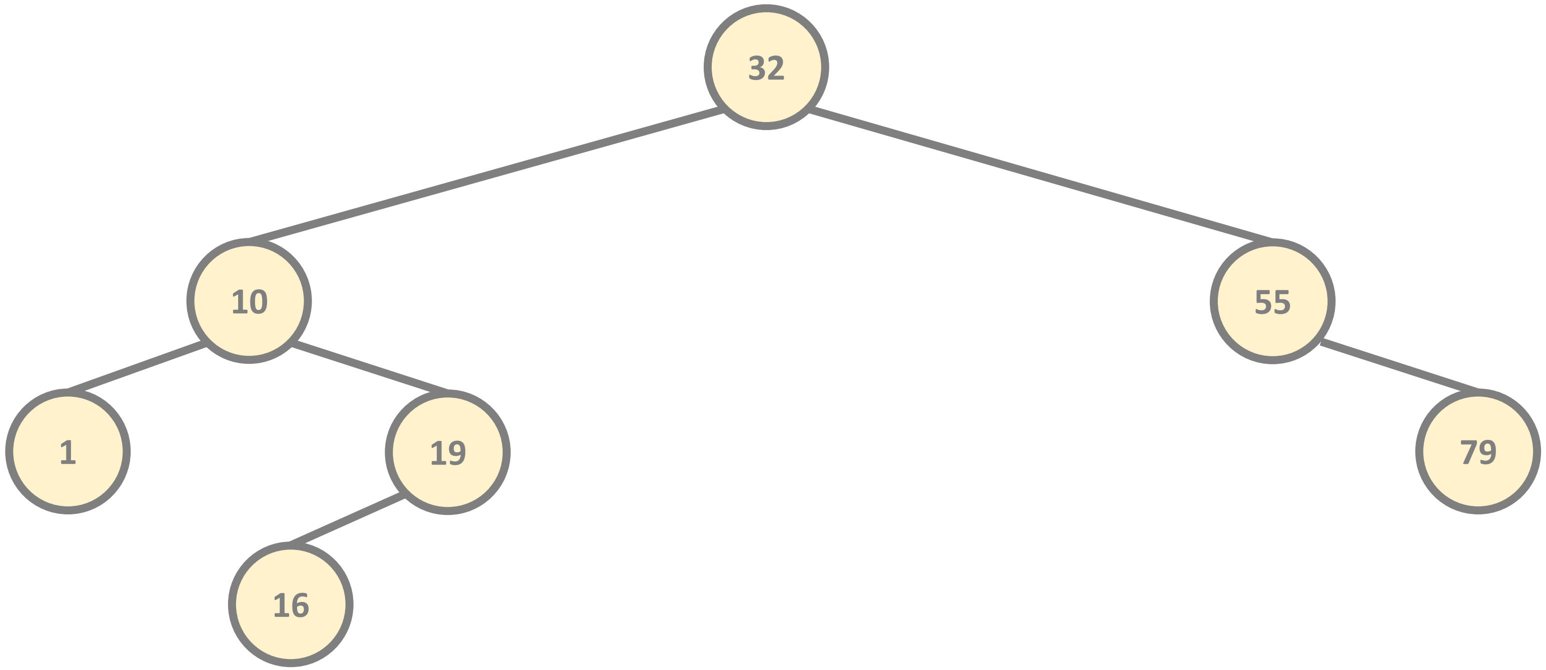
1.) REMOVING A LEAF NODE



basically we just have to notify the parent that the child has been removed
*- the node will be removed by the **garbage collector** -*

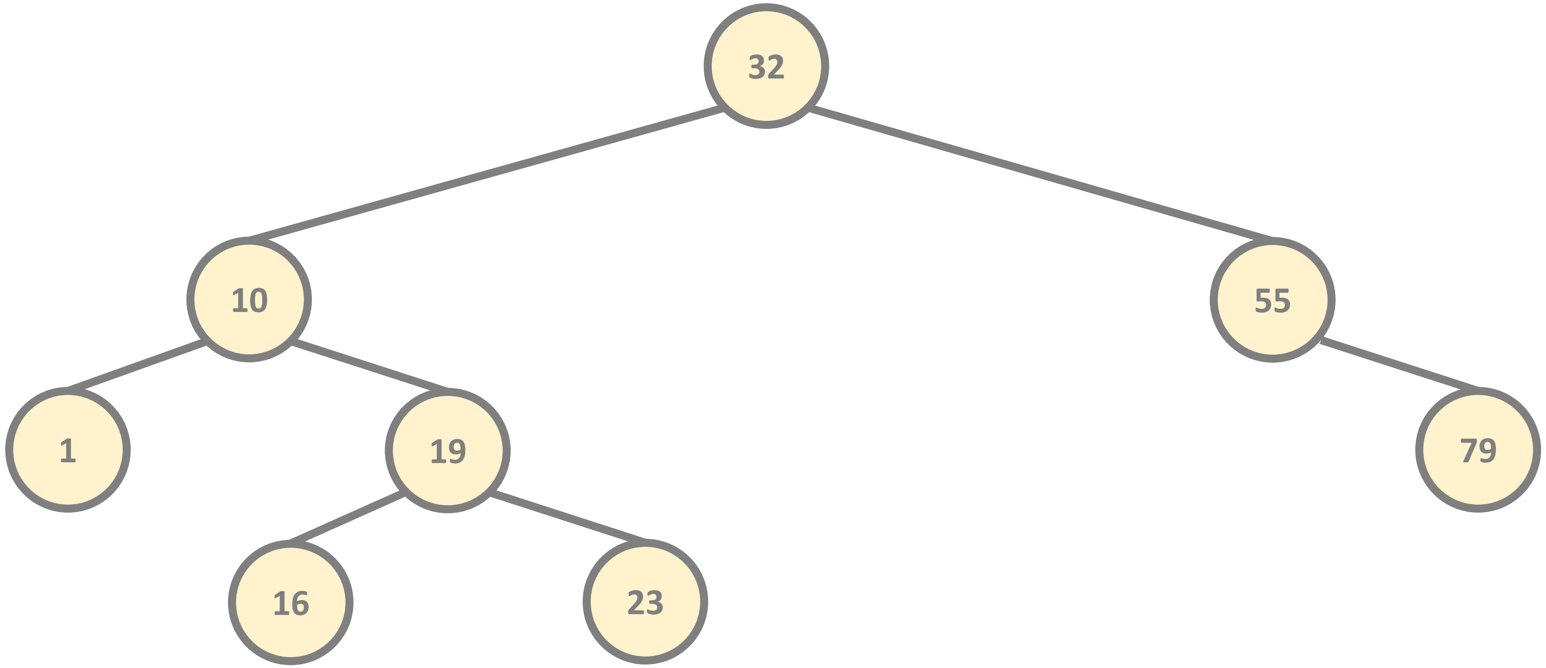
Binary Search Trees

1.) REMOVING A LEAF NODE



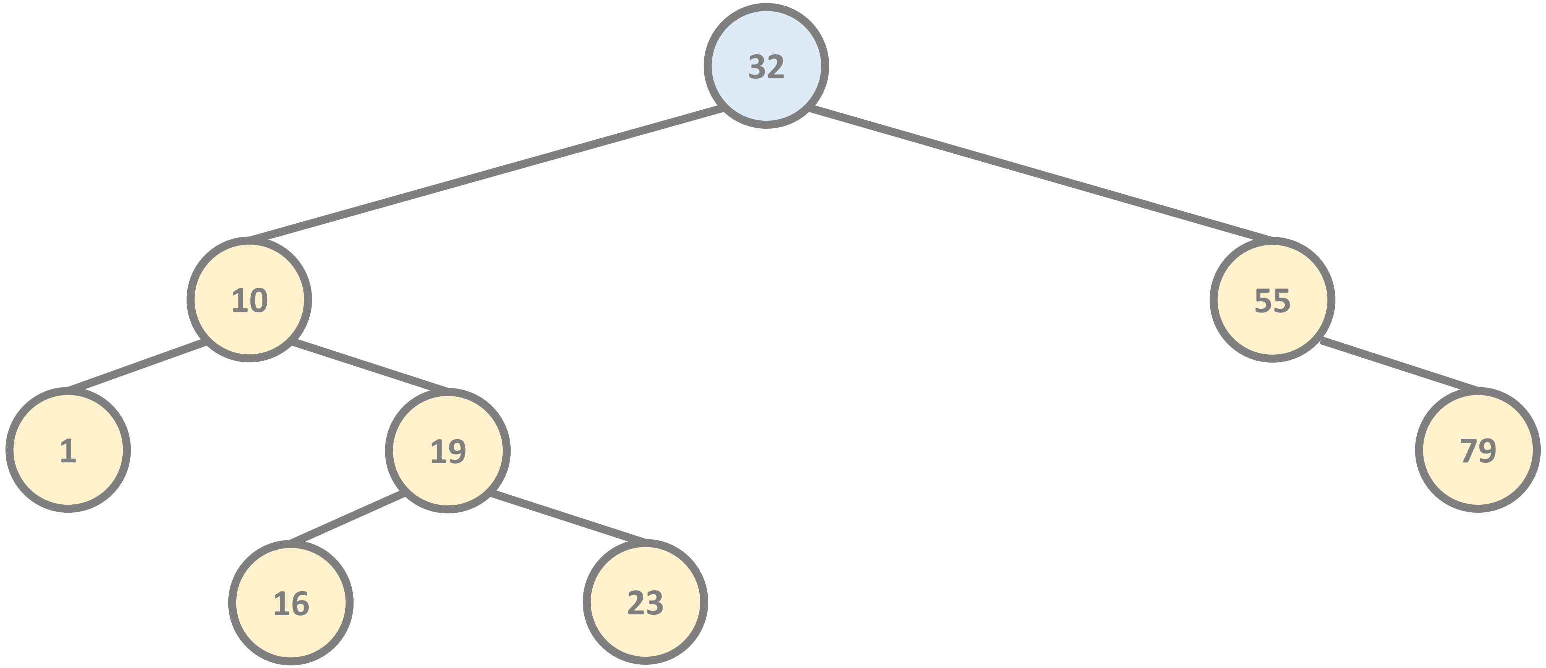
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



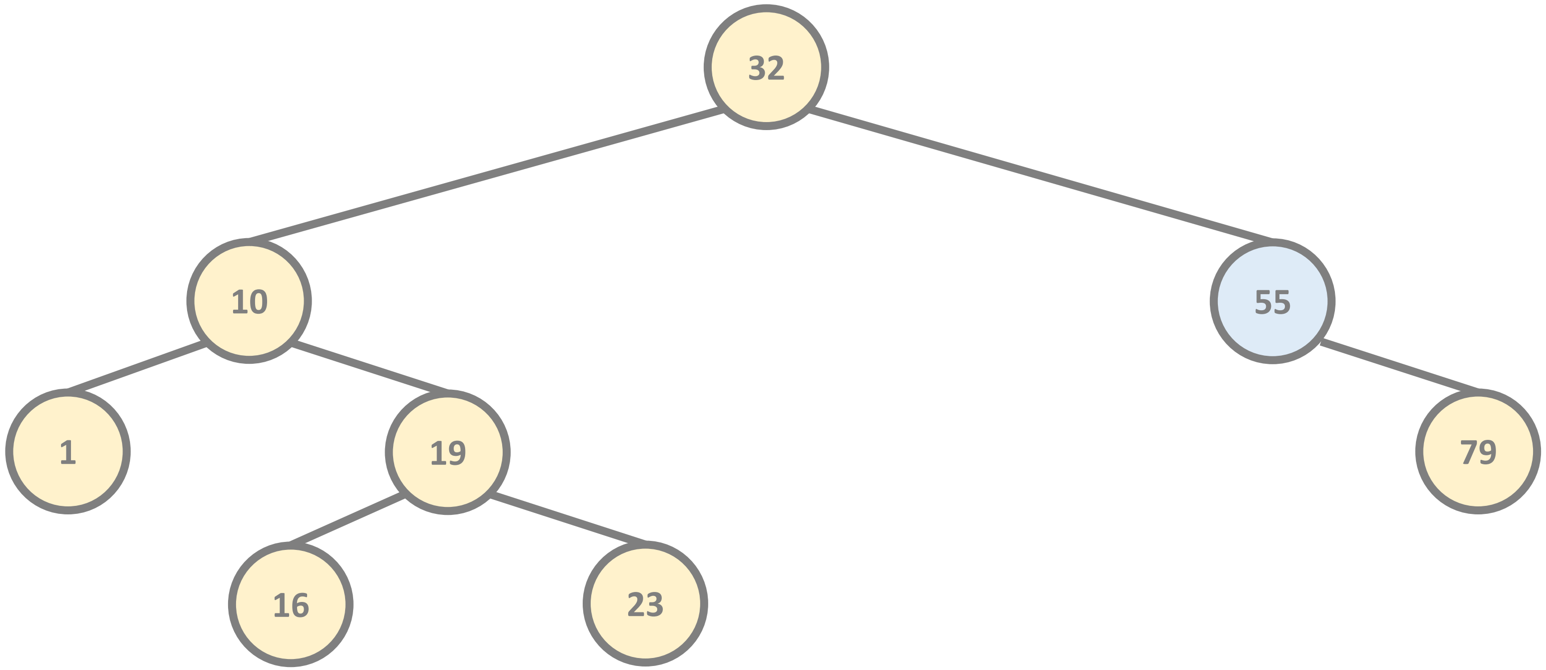
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



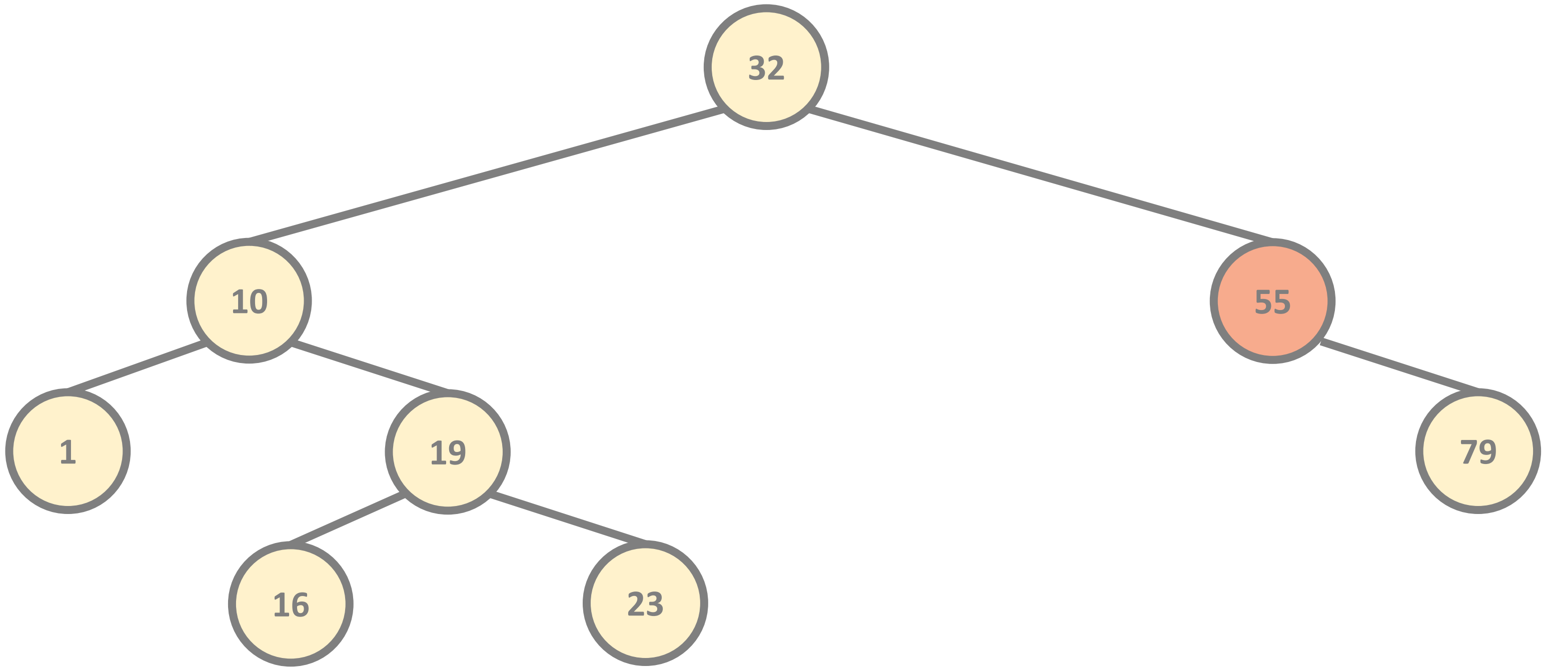
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



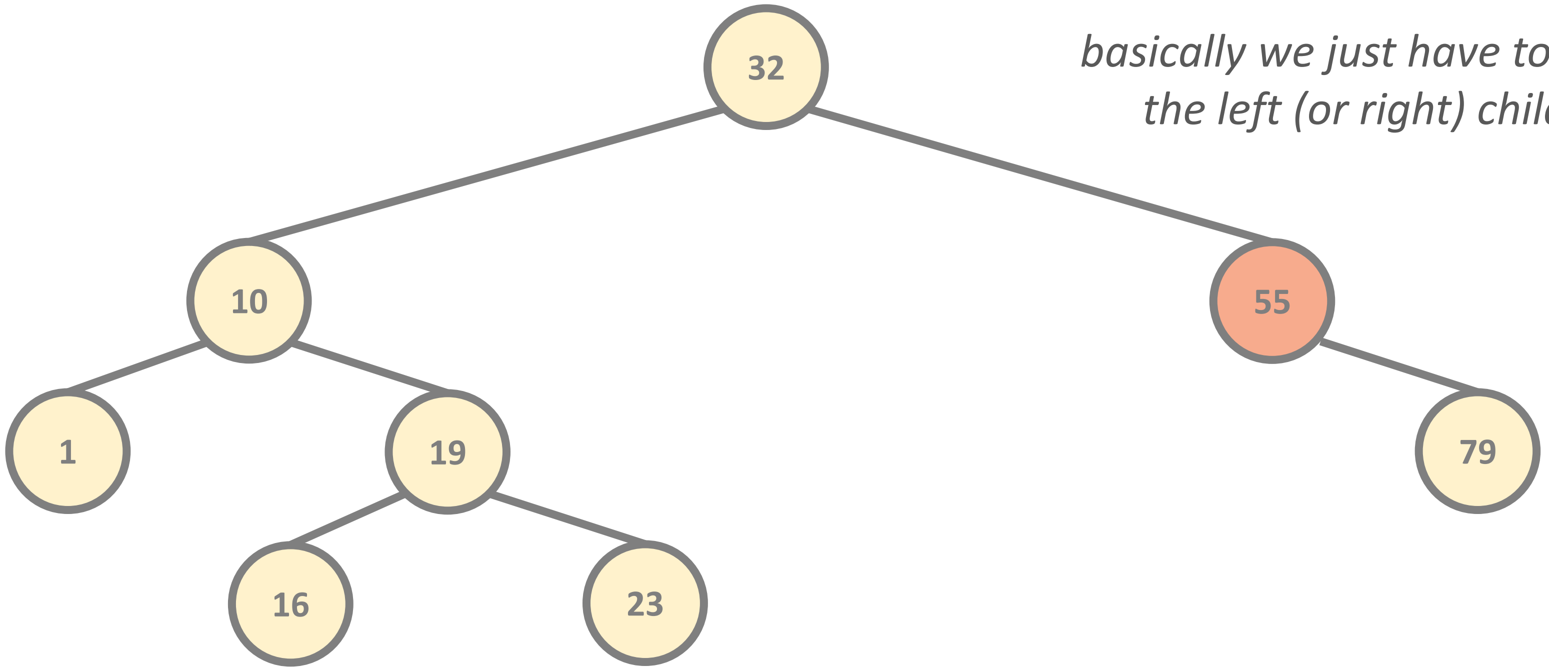
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



Binary Search Trees

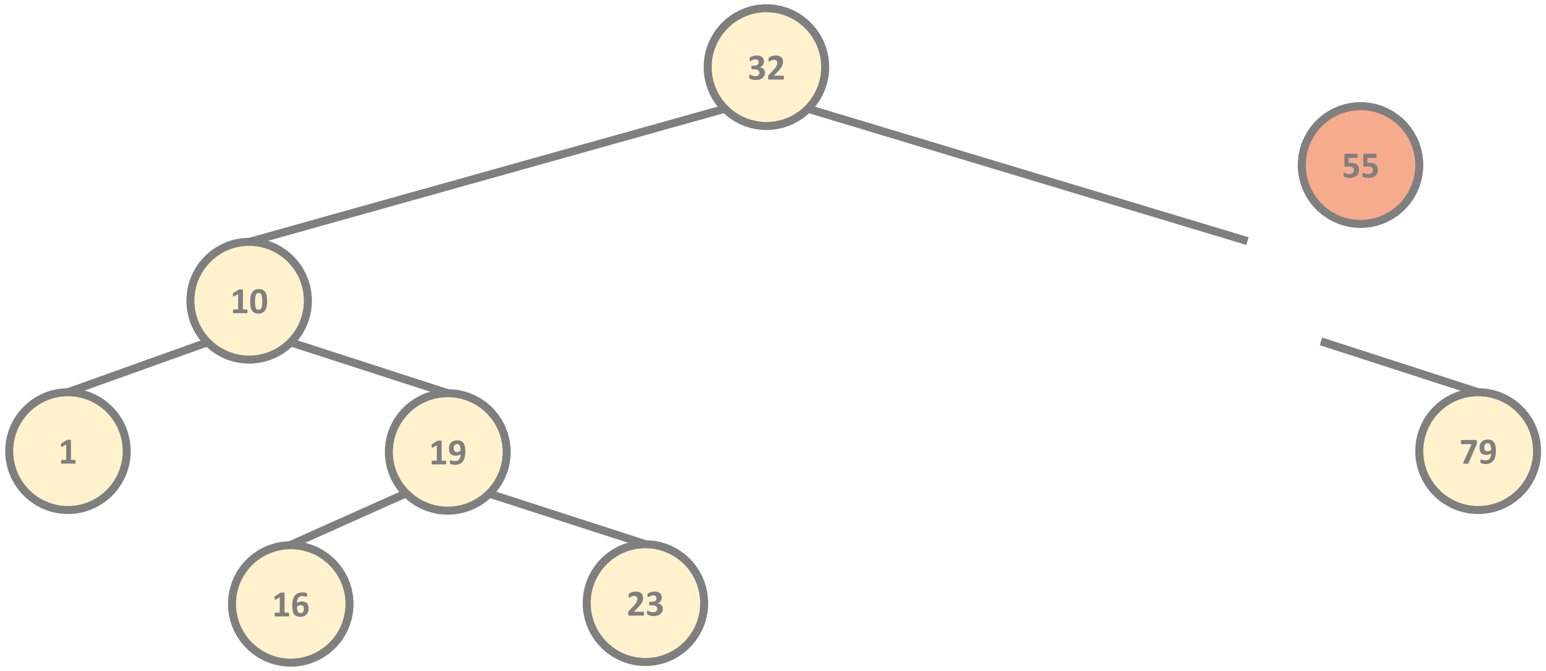
2.) REMOVING A NODE WITH A SINGLE CHILD



*basically we just have to **notify the parent** that the left (or right) child has been changed*

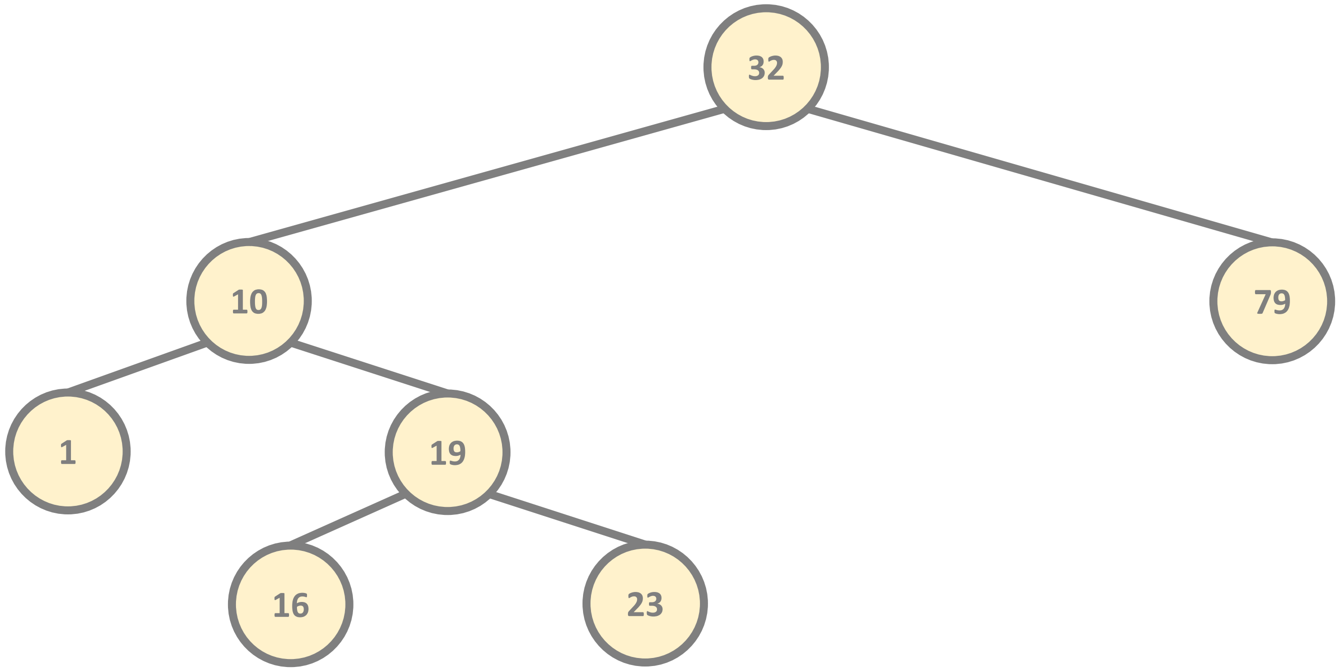
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



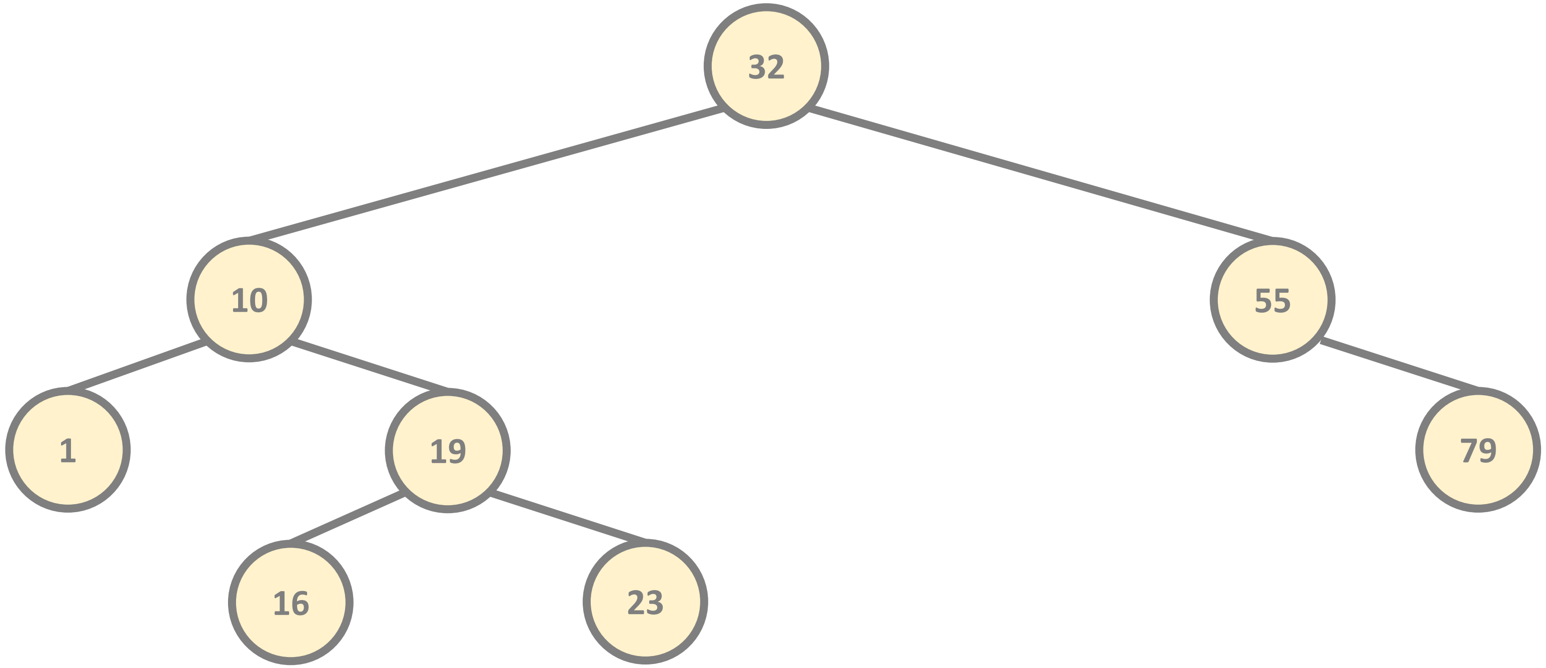
Binary Search Trees

2.) REMOVING A NODE WITH A SINGLE CHILD



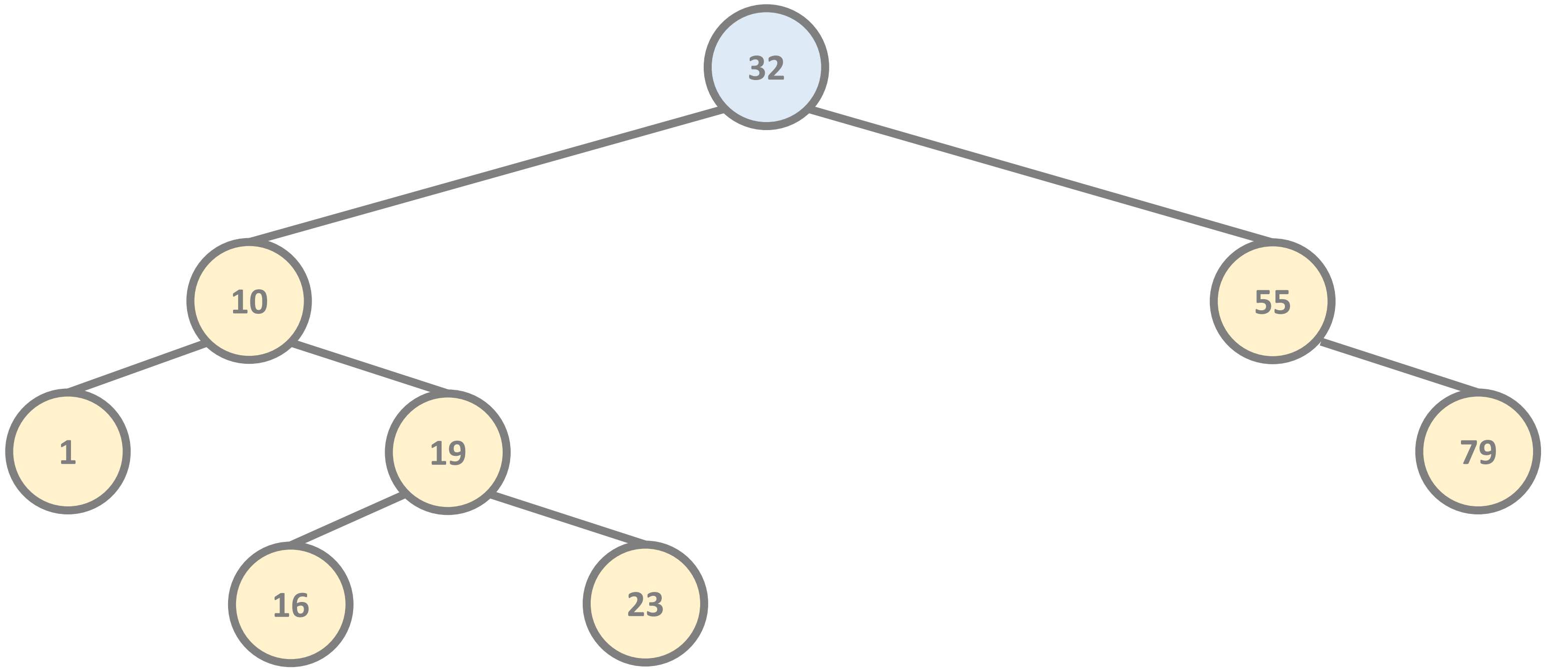
Binary Search Trees

3.) REMOVING A NODE WITH TWO CHILDREN



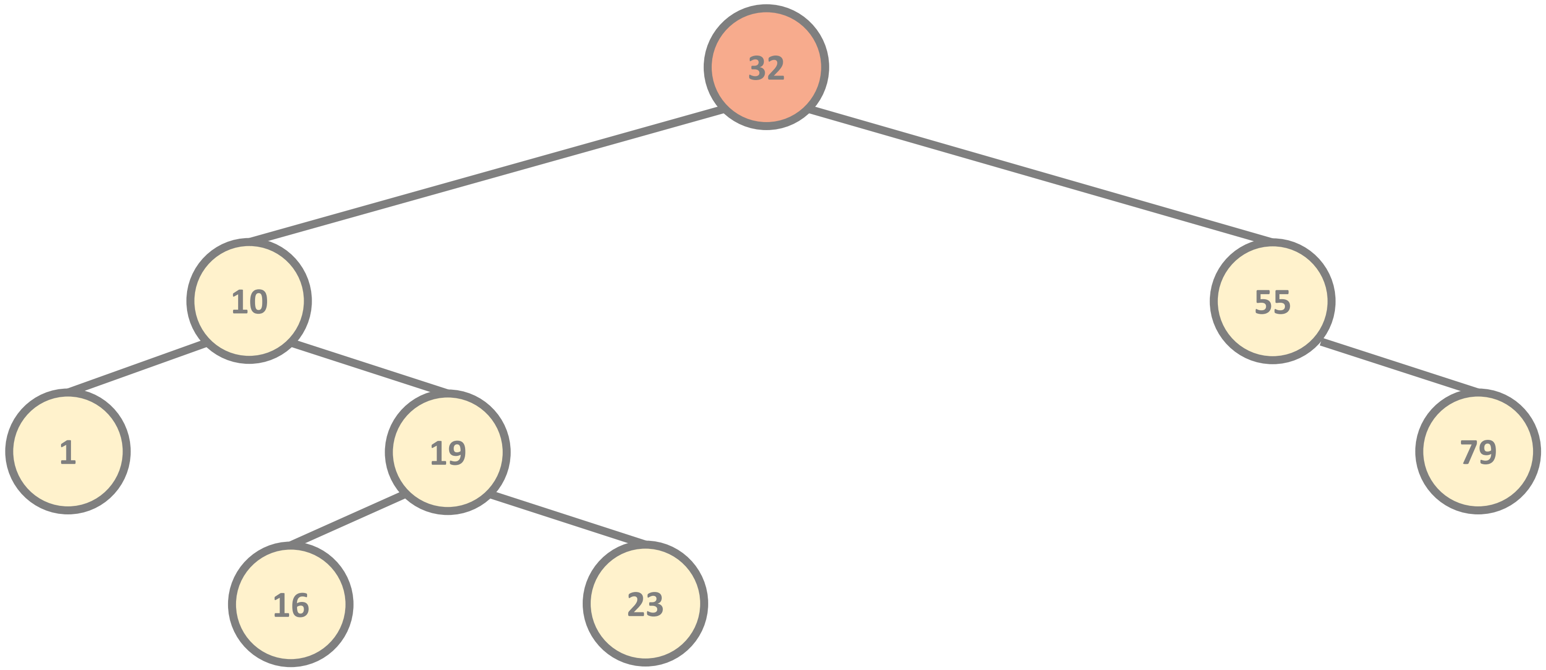
Binary Search Trees

3.) REMOVING A NODE WITH TWO CHILDREN



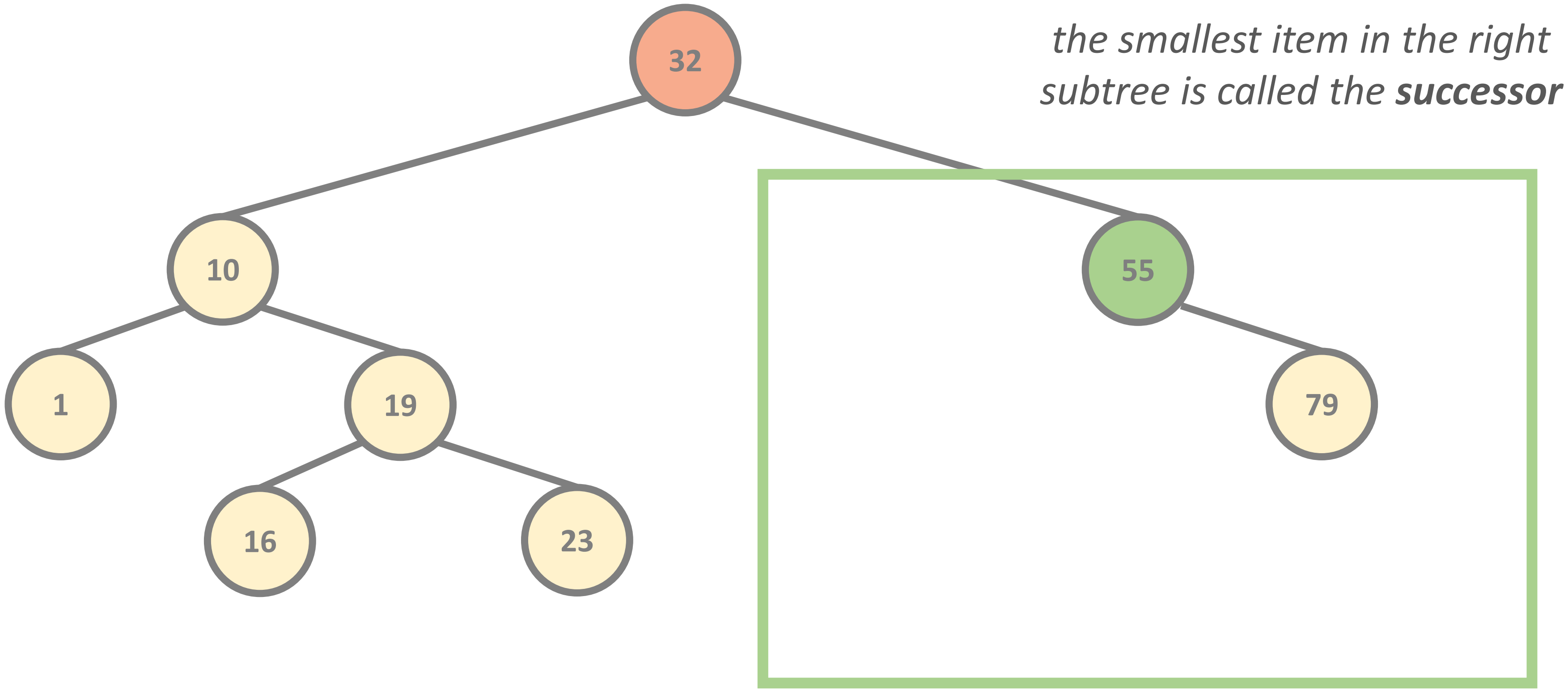
Binary Search Trees

3.) REMOVING A NODE WITH TWO CHILDREN



Binary Search Trees

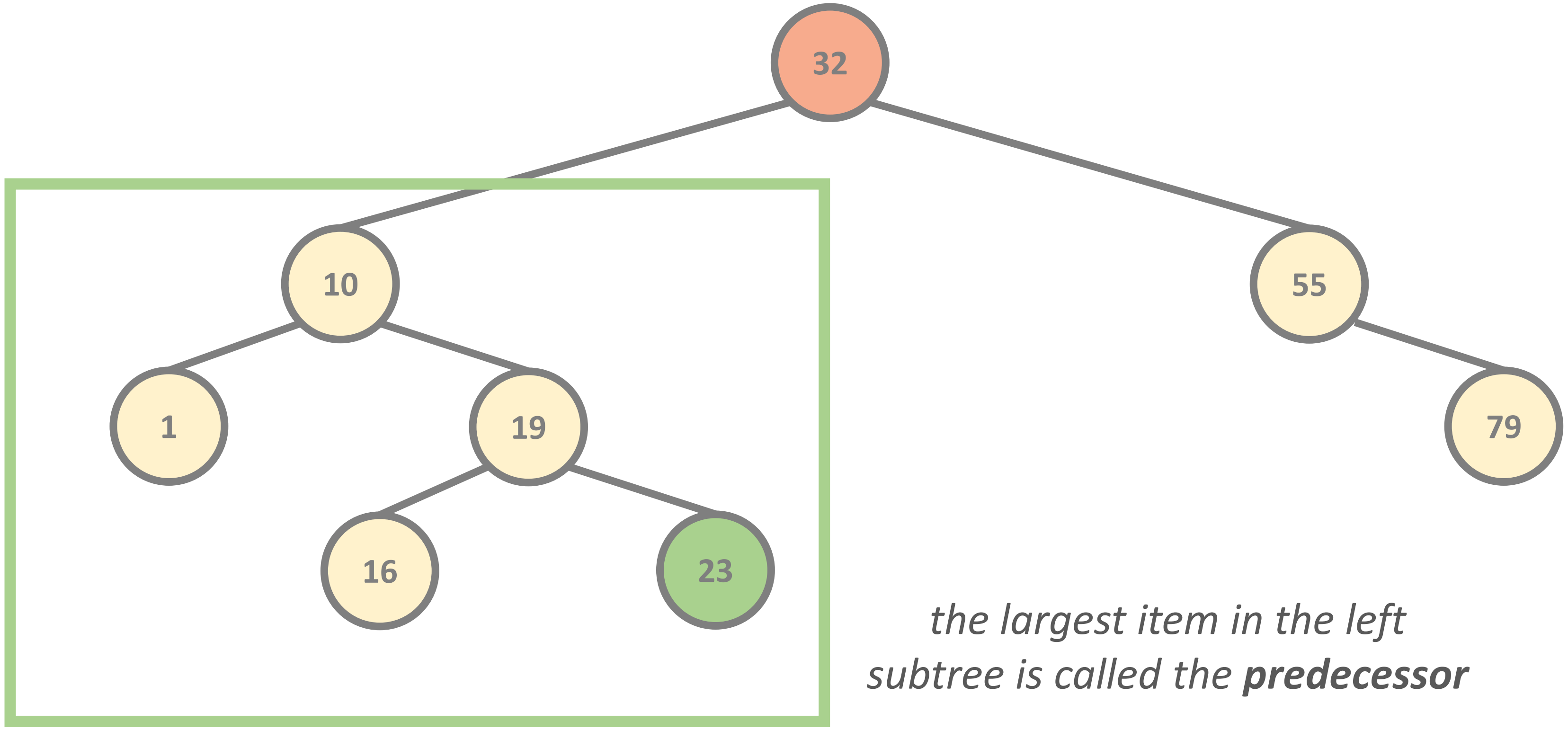
3.) REMOVING A NODE WITH TWO CHILDREN



*the smallest item in the right subtree is called the **successor***

Binary Search Trees

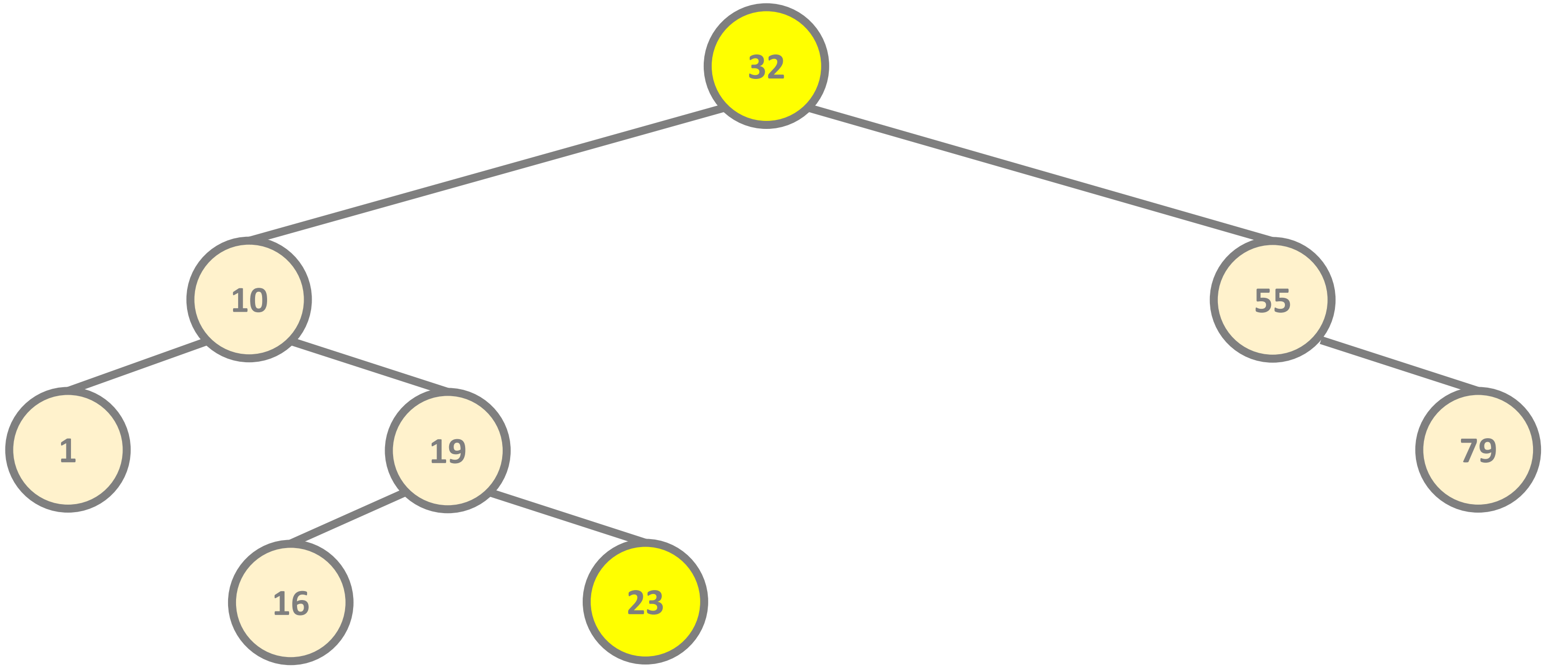
3.) REMOVING A NODE WITH TWO CHILDREN



*the largest item in the left subtree is called the **predecessor***

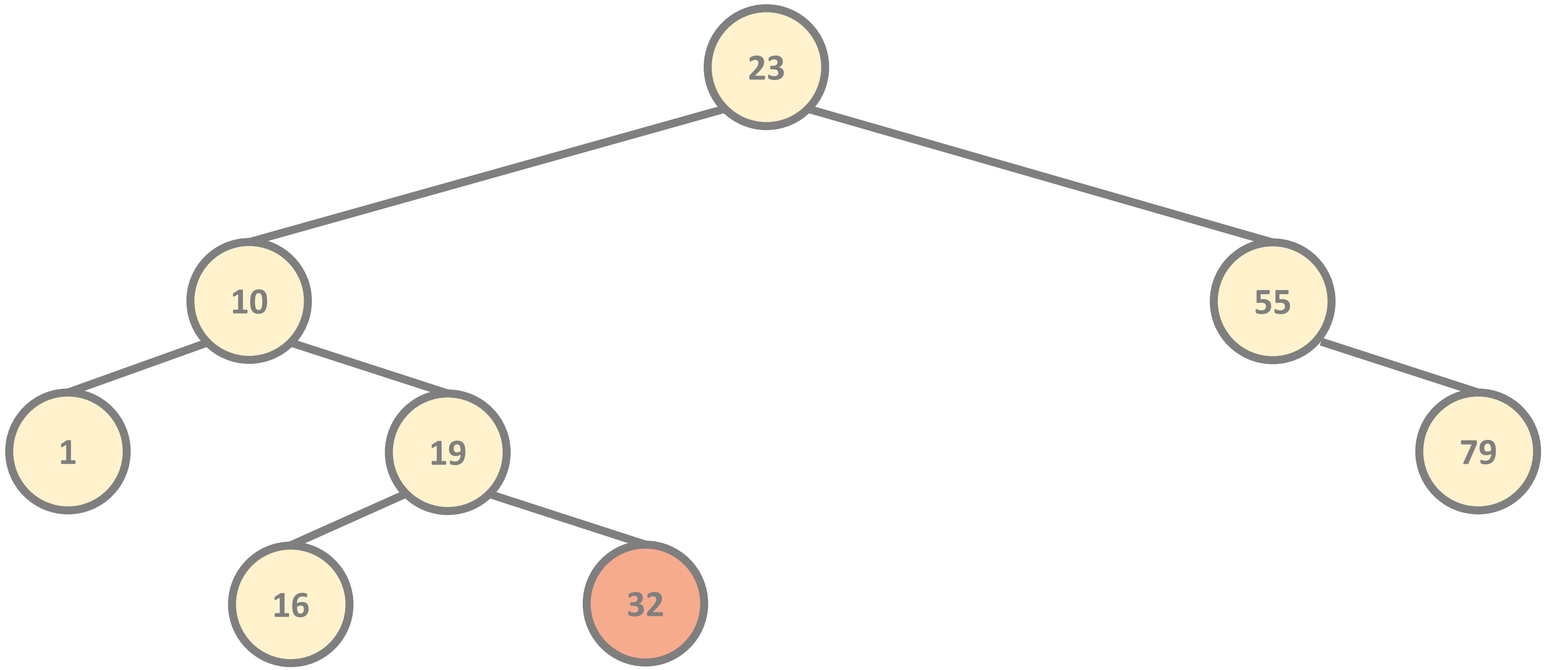
Binary Search Trees

3.) REMOVING A NODE WITH TWO CHILDREN



Binary Search Trees

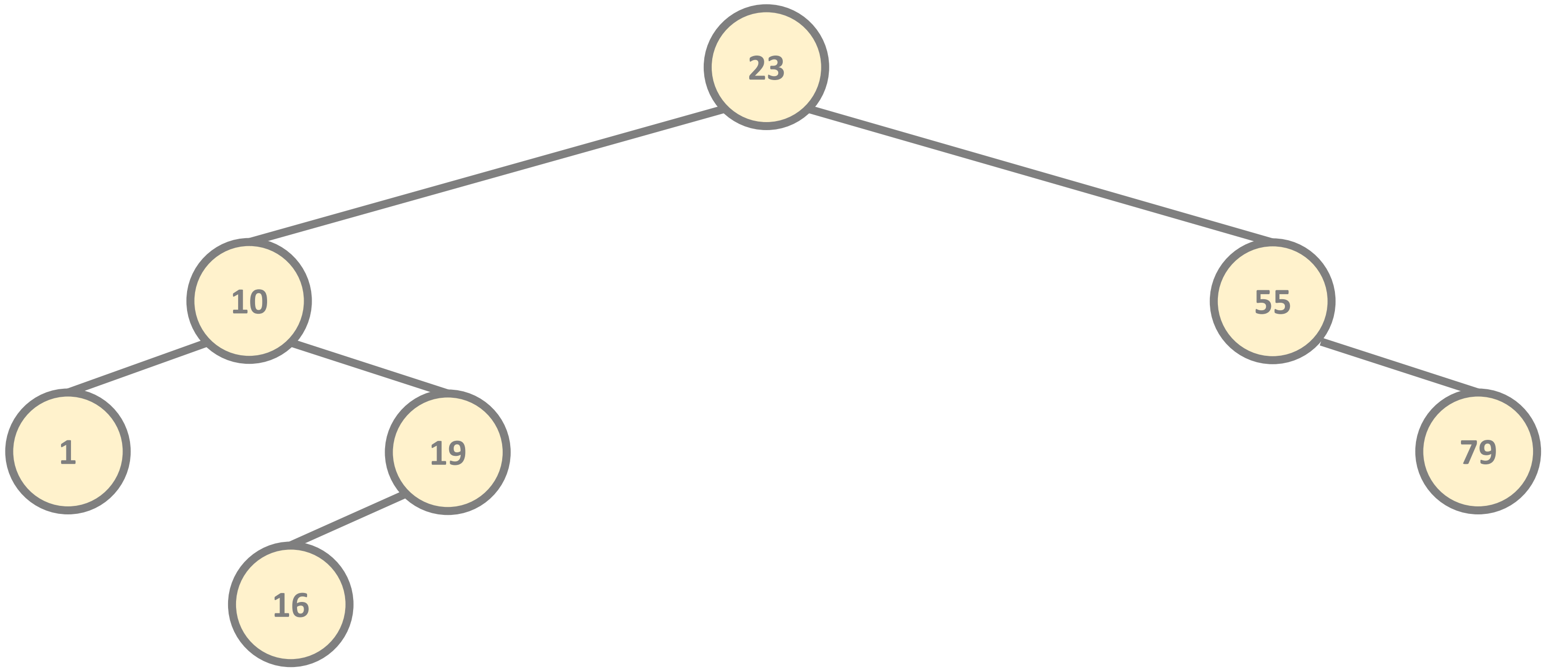
3.) REMOVING A NODE WITH TWO CHILDREN



we know how to deal with leaf nodes
*(**mathematical reduction**)*

Binary Search Trees

3.) REMOVING A NODE WITH TWO CHILDREN



Binary Search Trees
(Algorithms and Data Structures)

Binary Search Tree Traversal

Tree traversal means visiting every node of the binary search tree exactly once in **O(N)** linear running time

1.) **pre-order traversal**

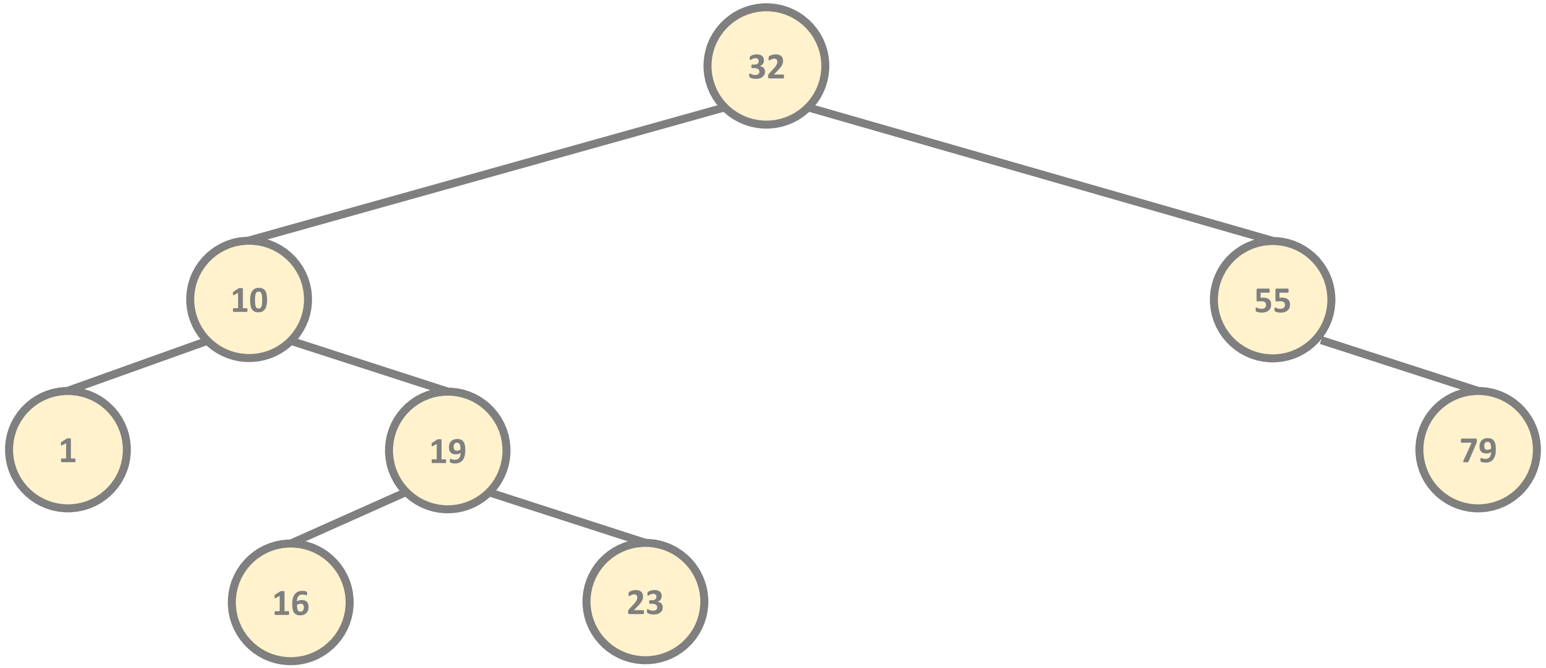
2.) **in-order traversal**

3.) **post-order traversal**

Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

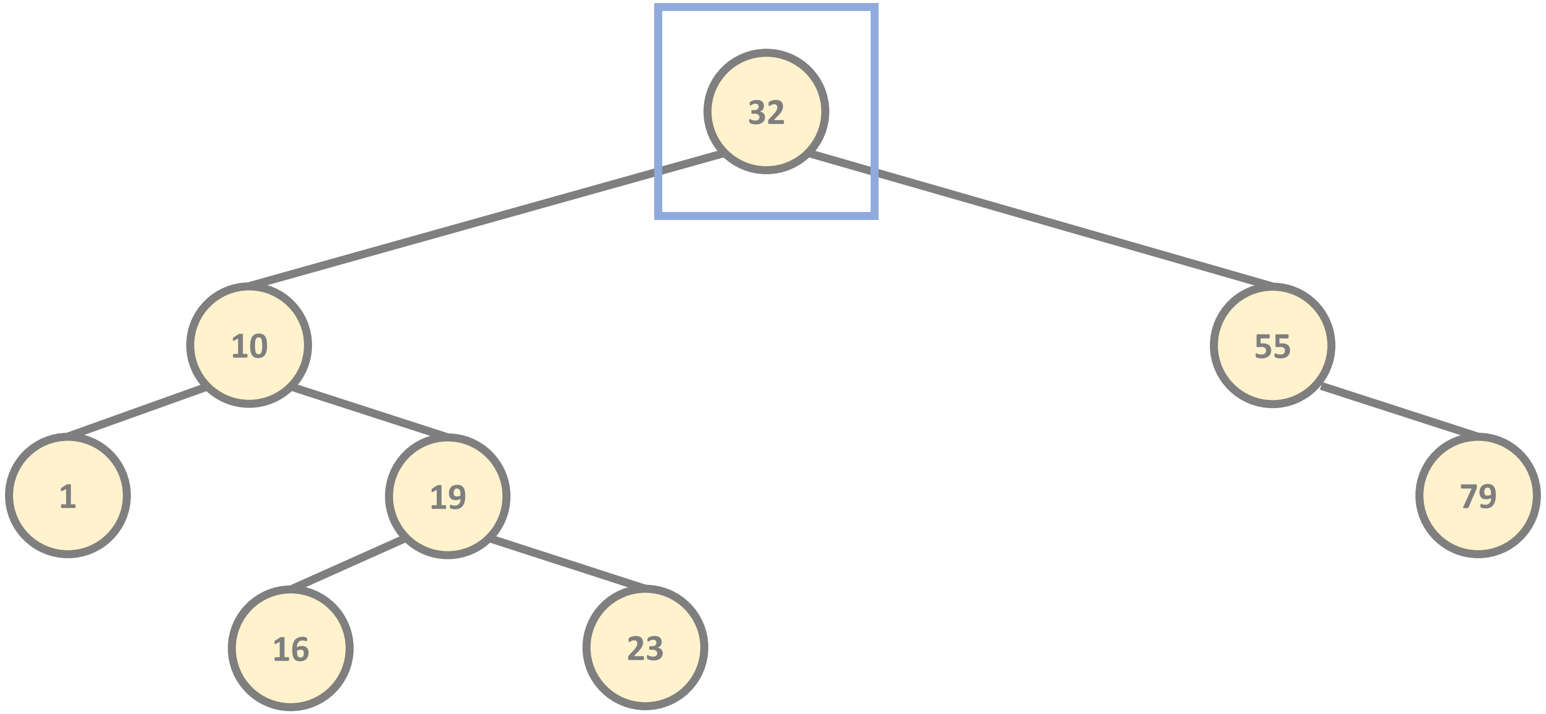
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

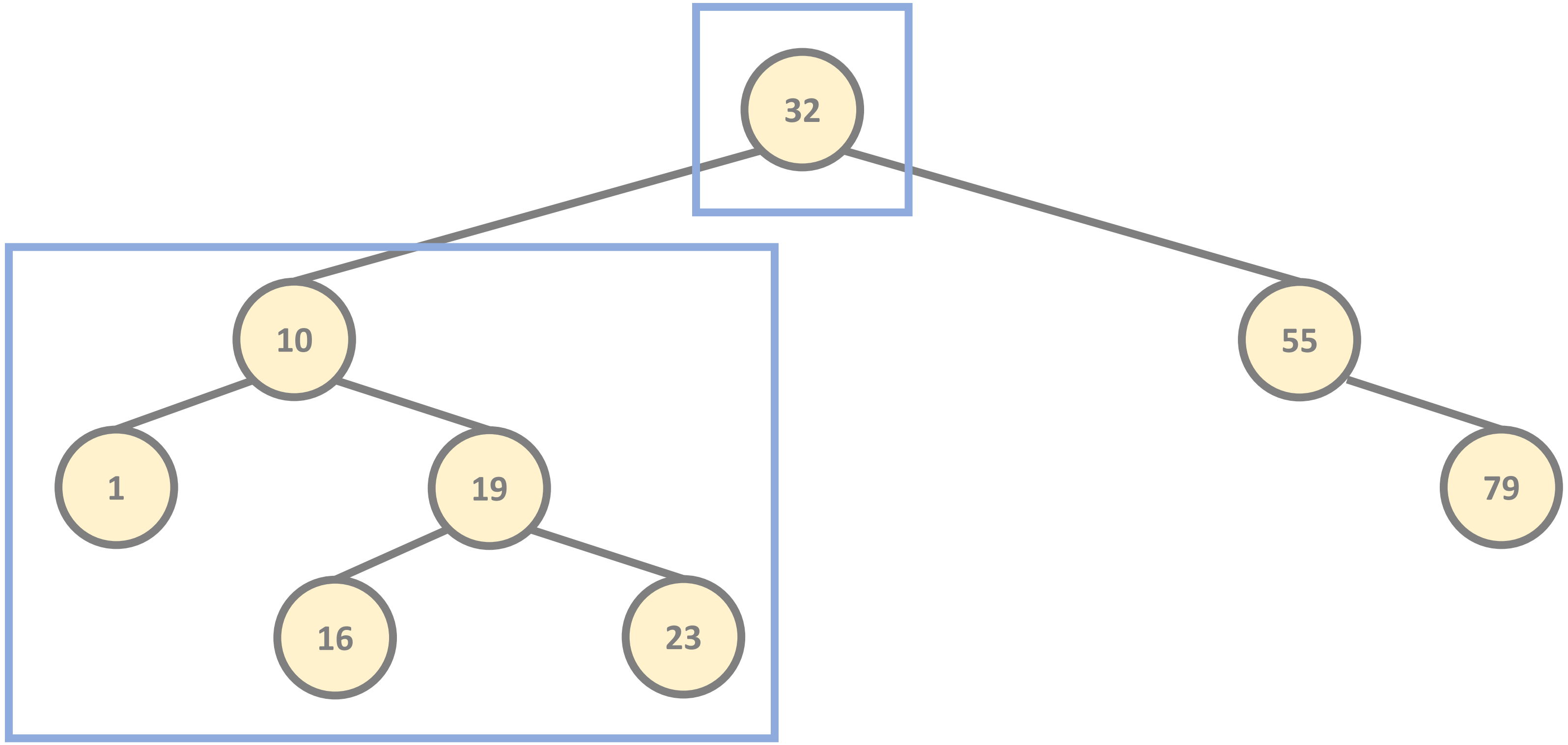
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

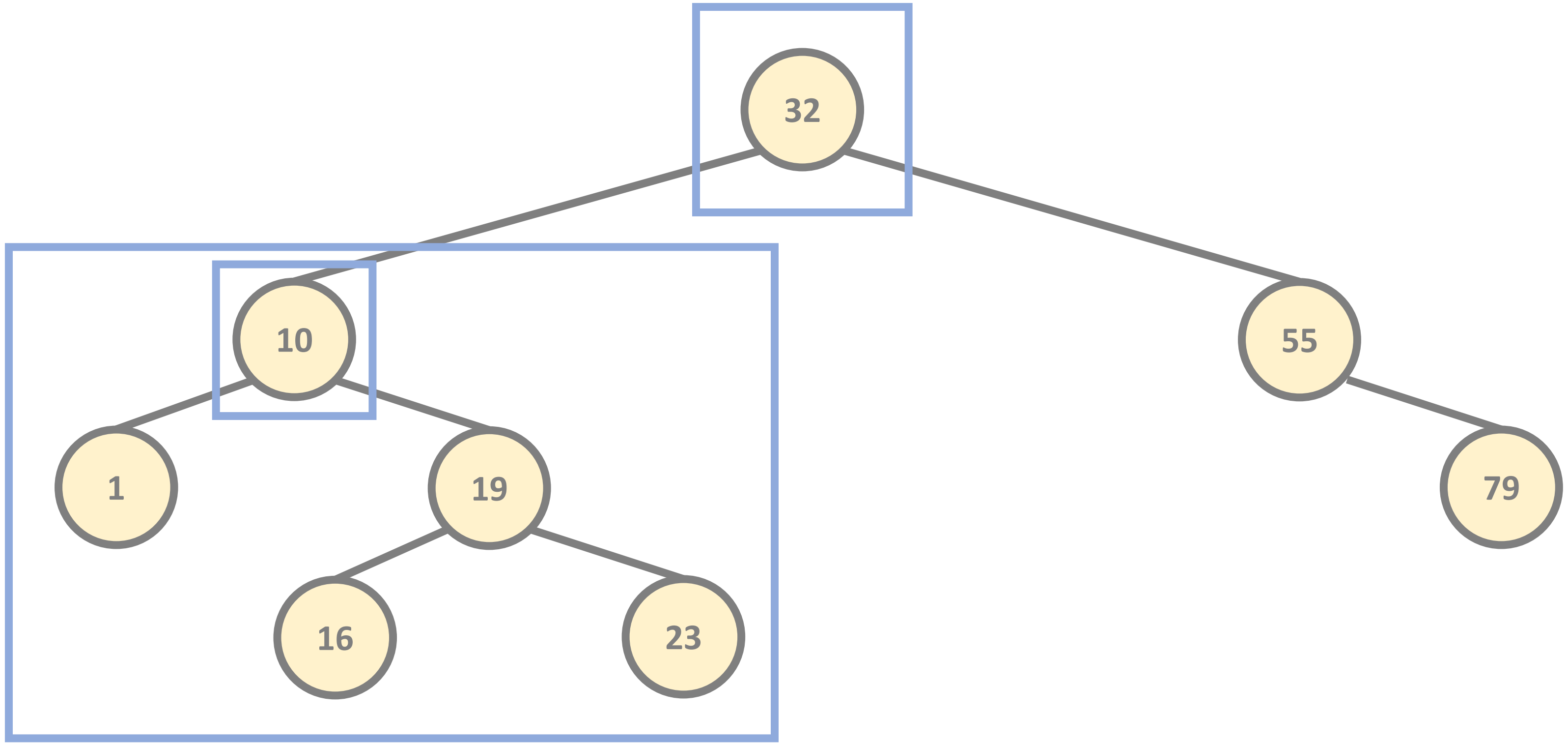
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

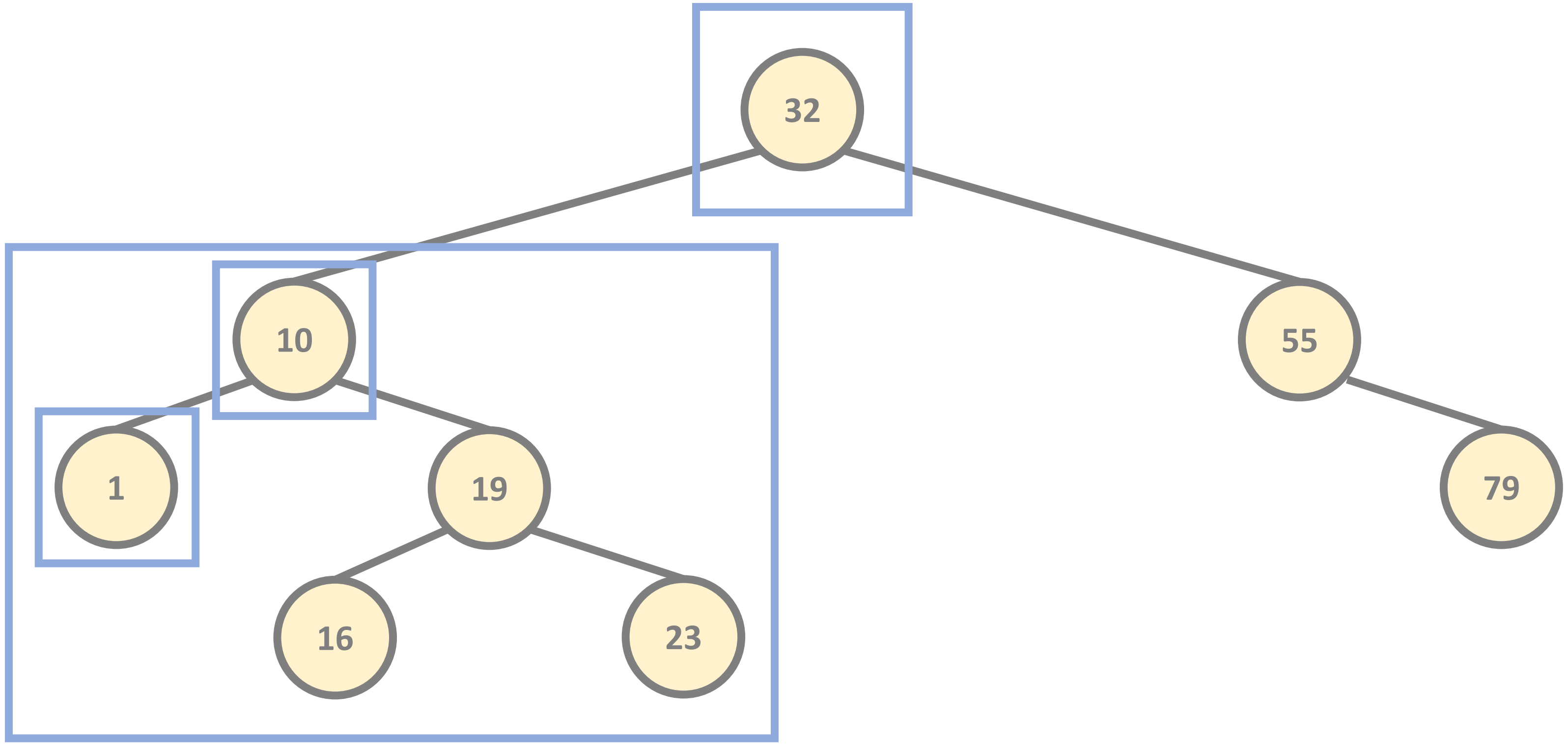
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

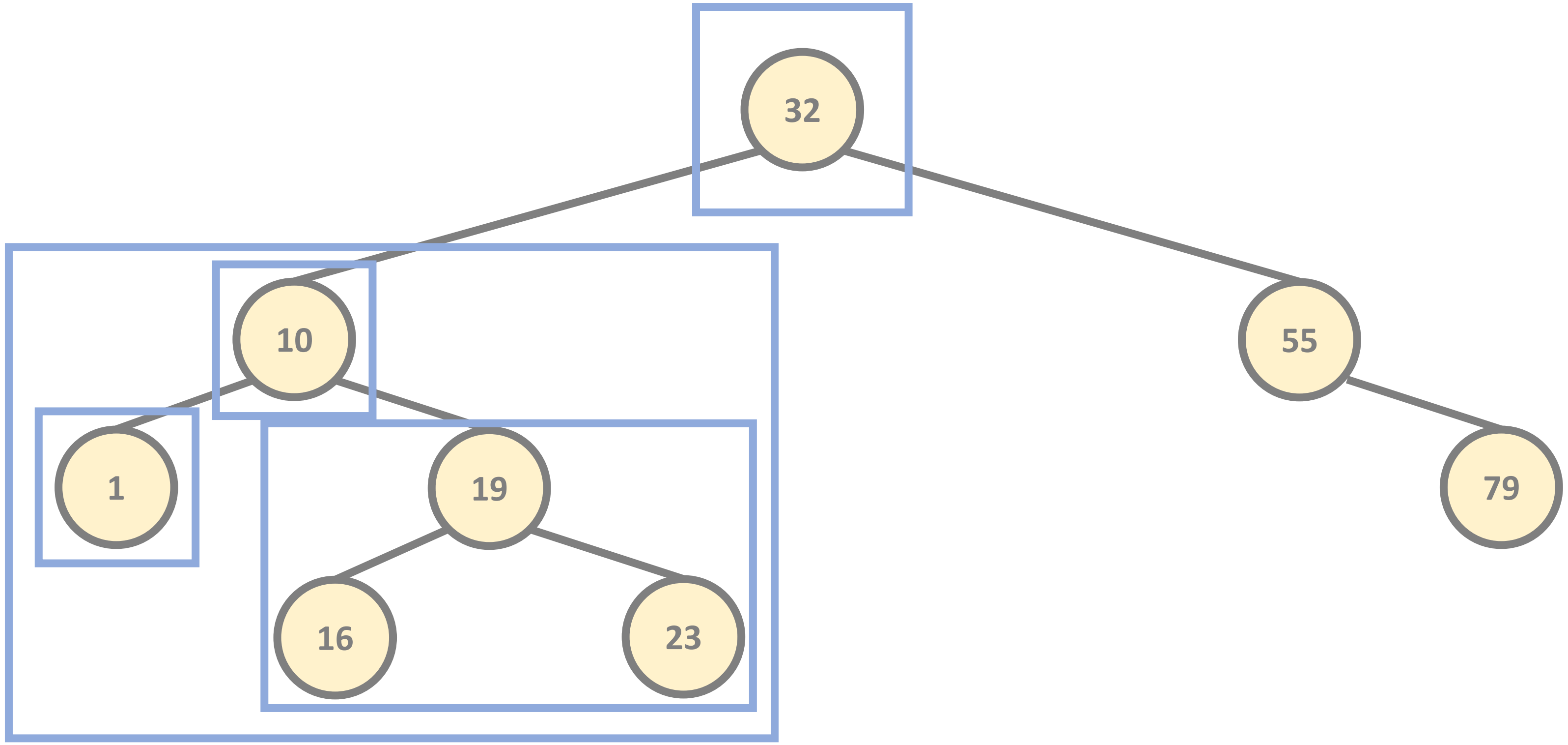
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

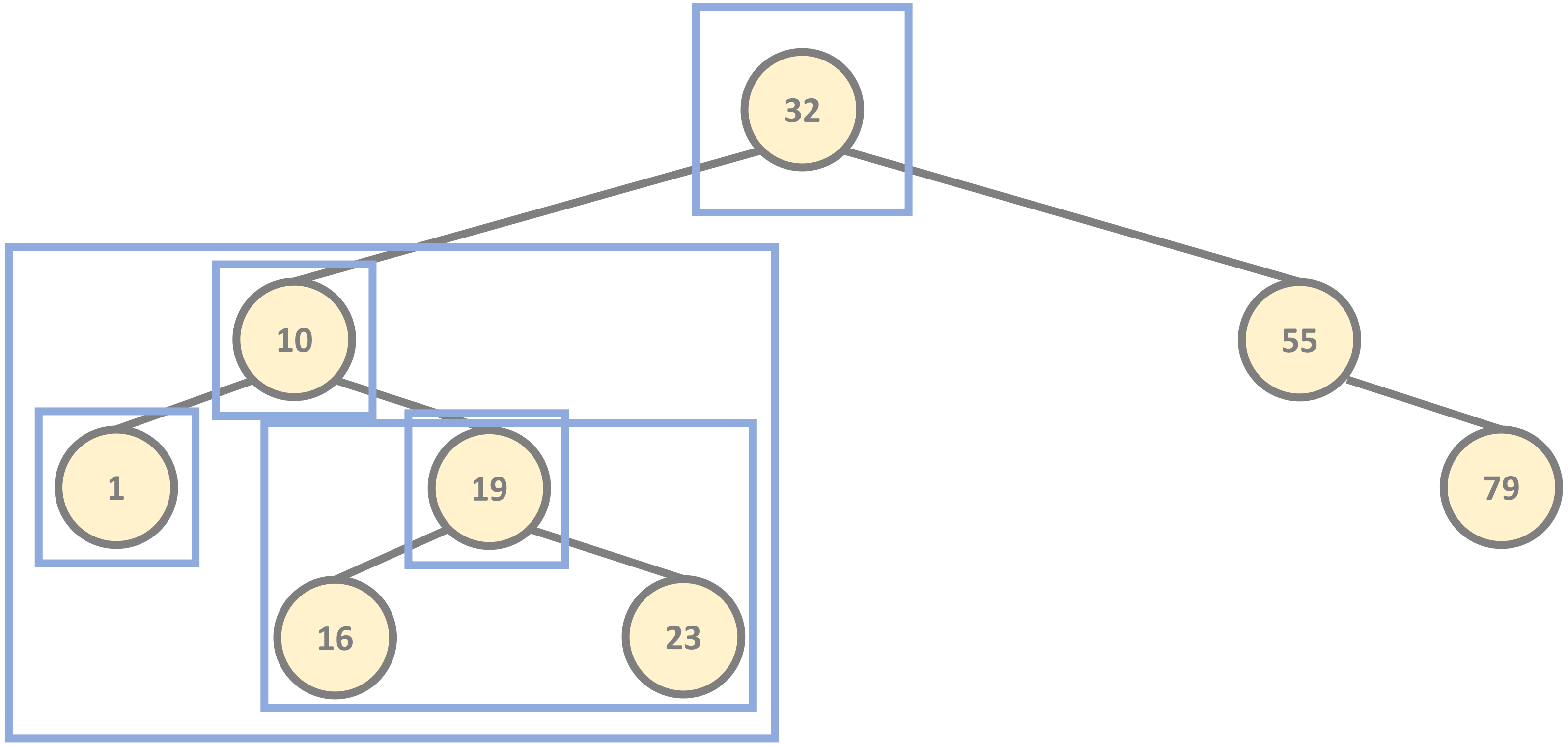
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

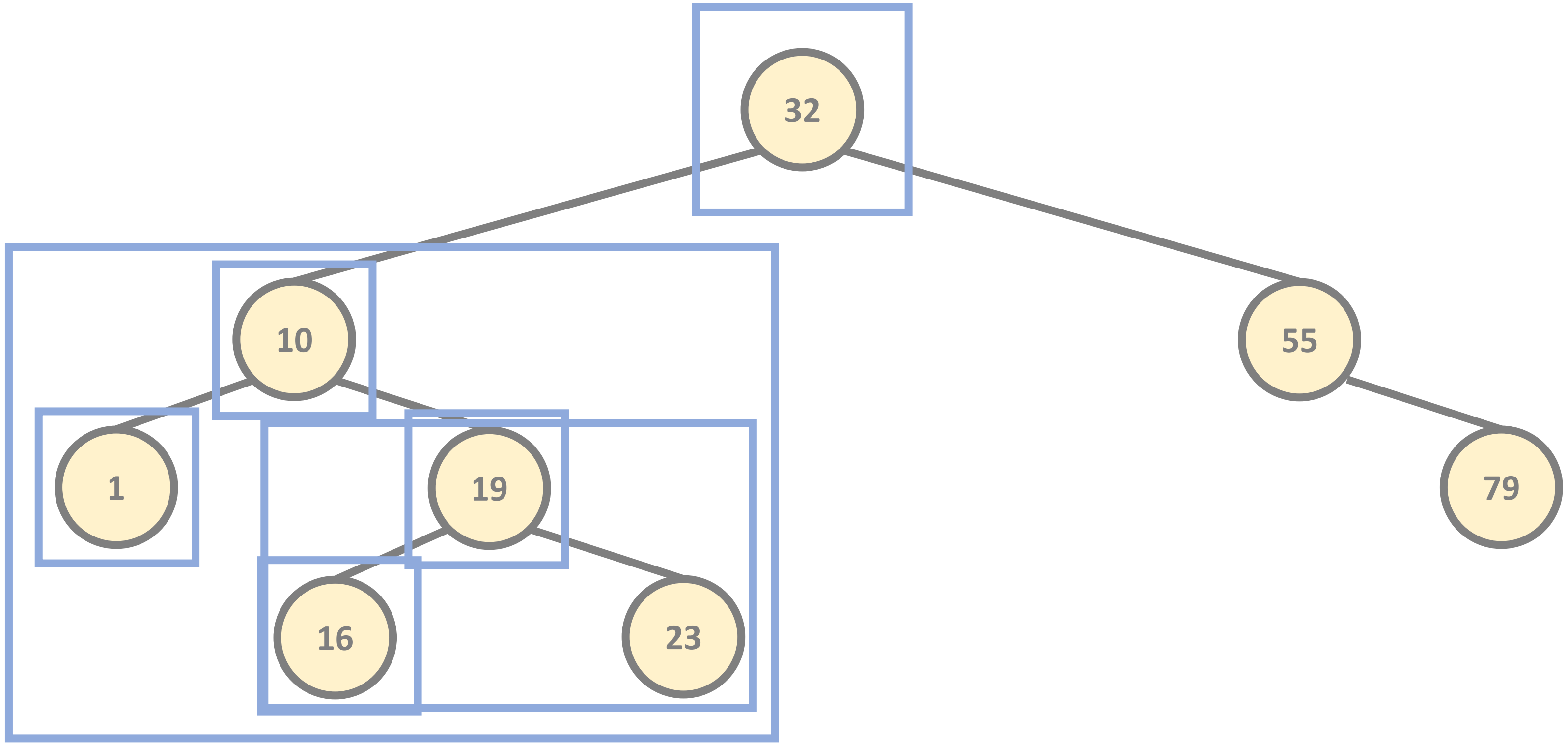
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

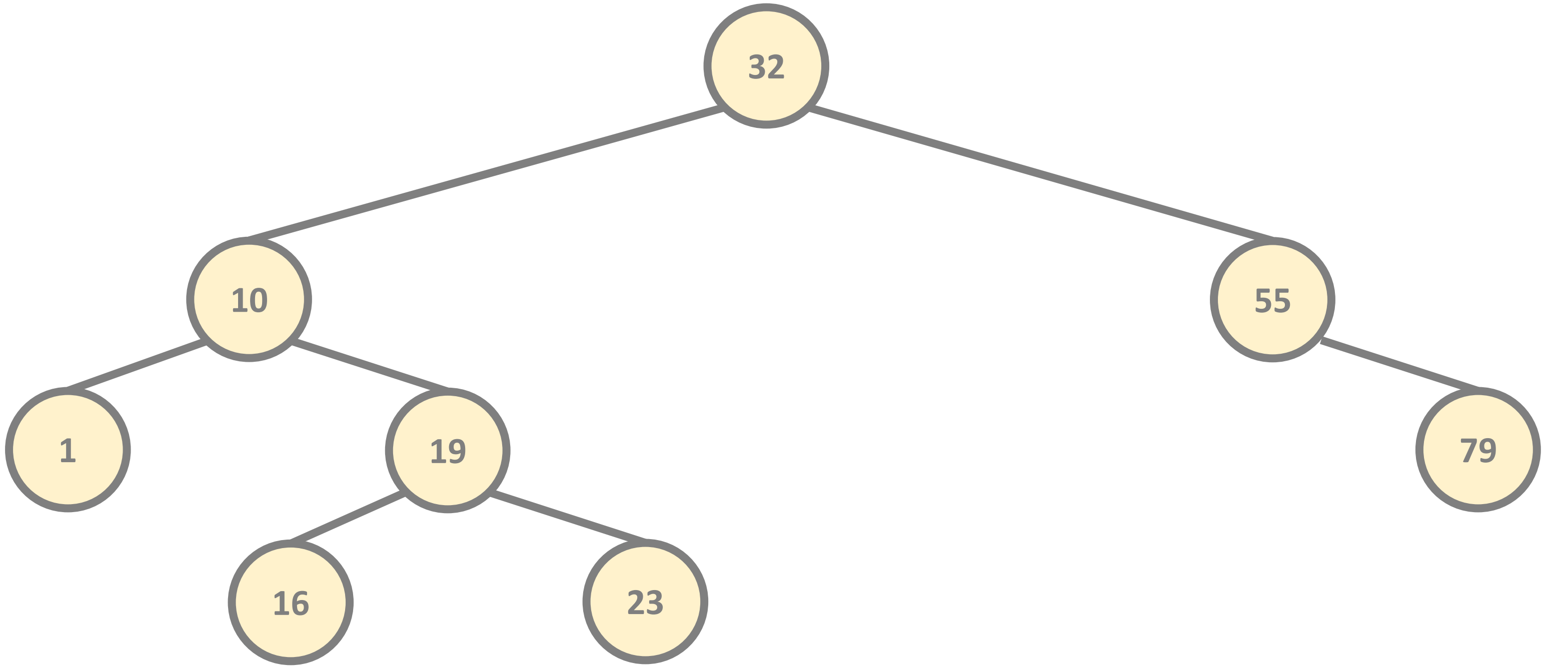
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

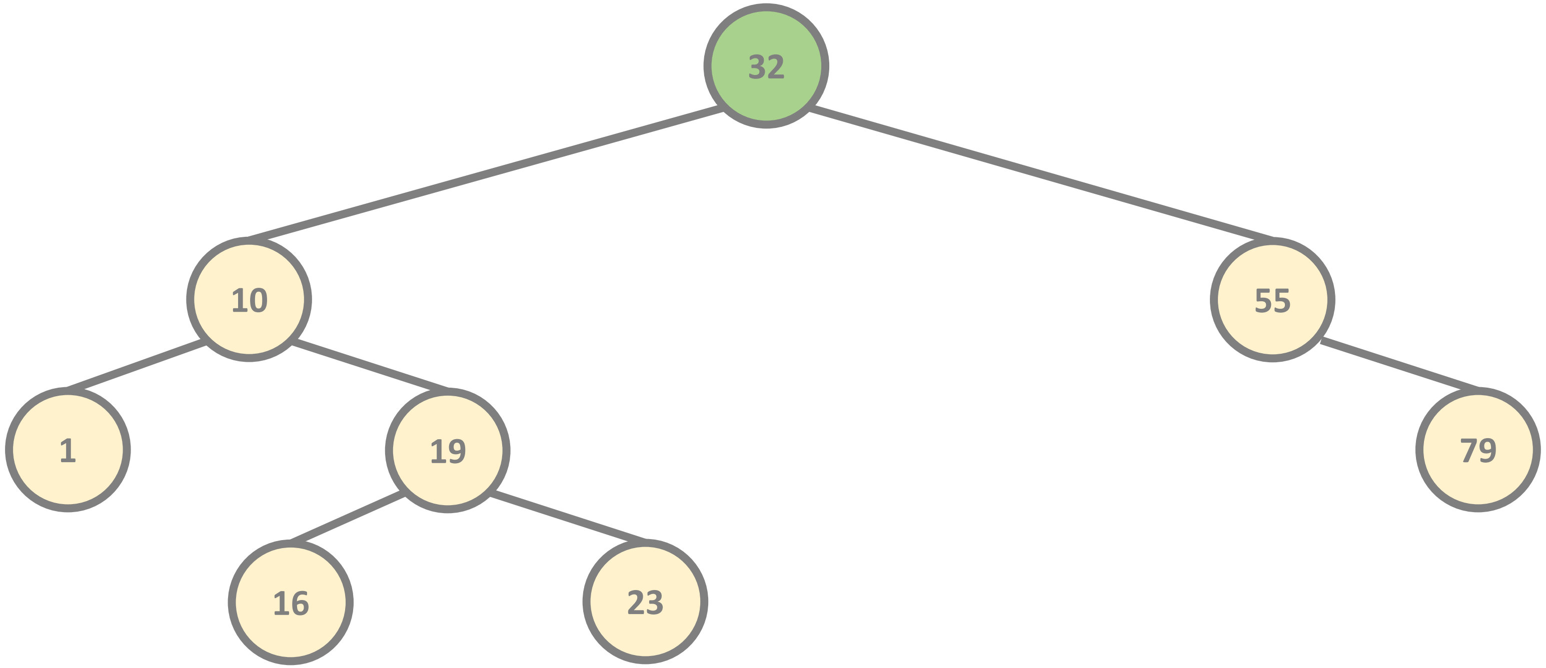
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

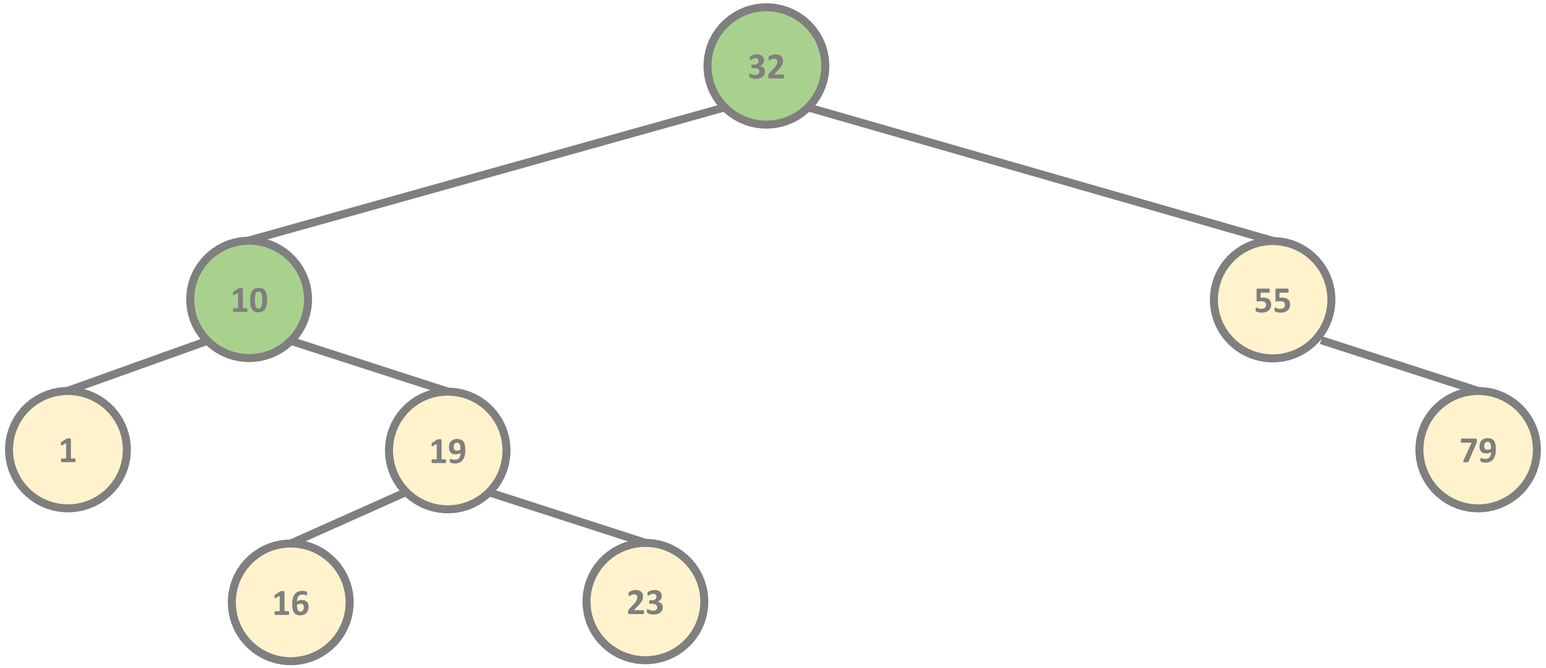
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

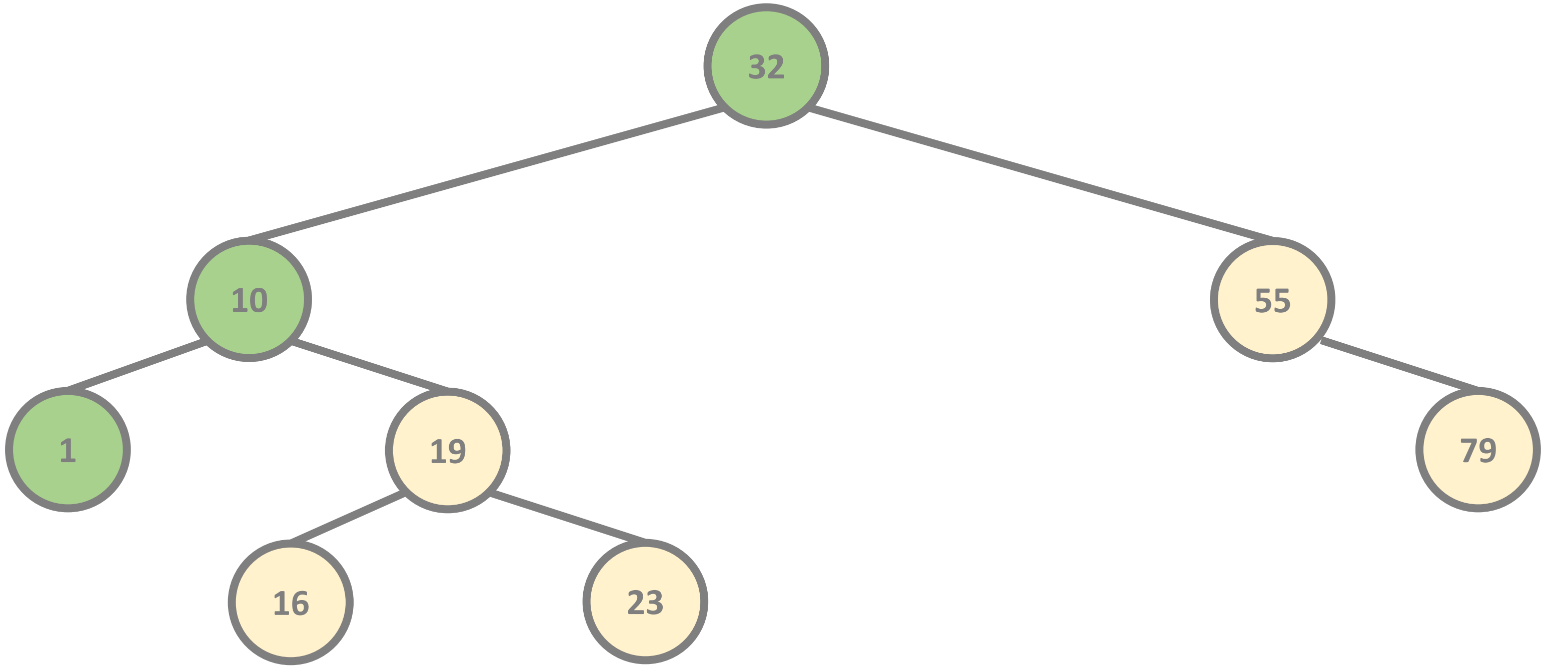
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

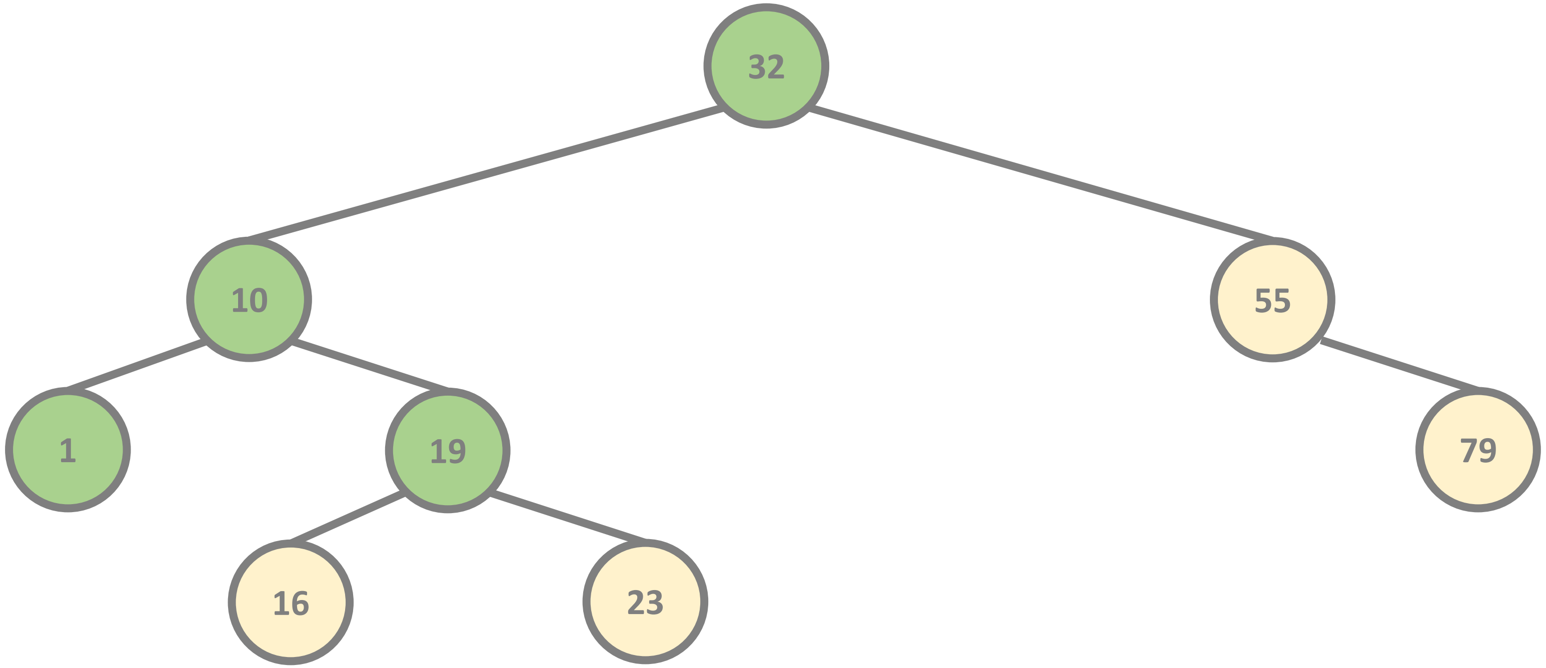
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

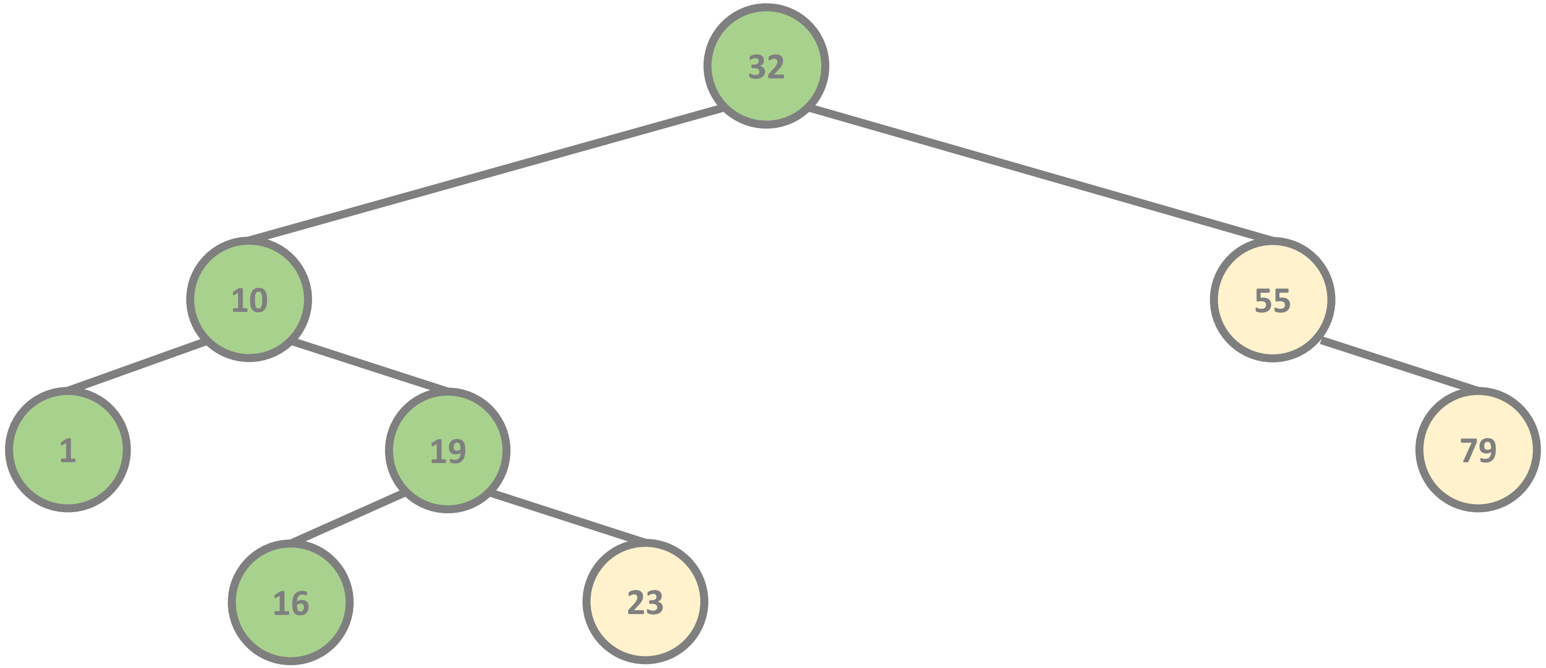
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

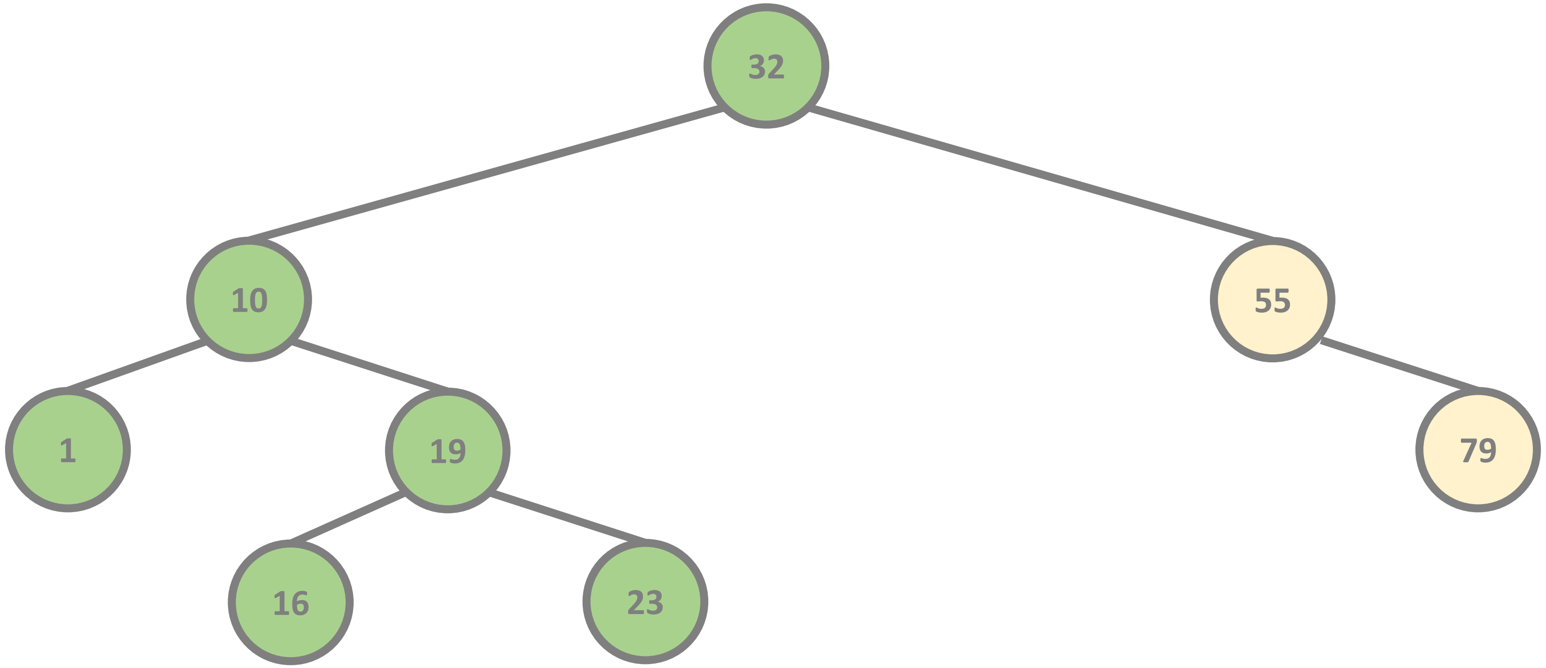
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

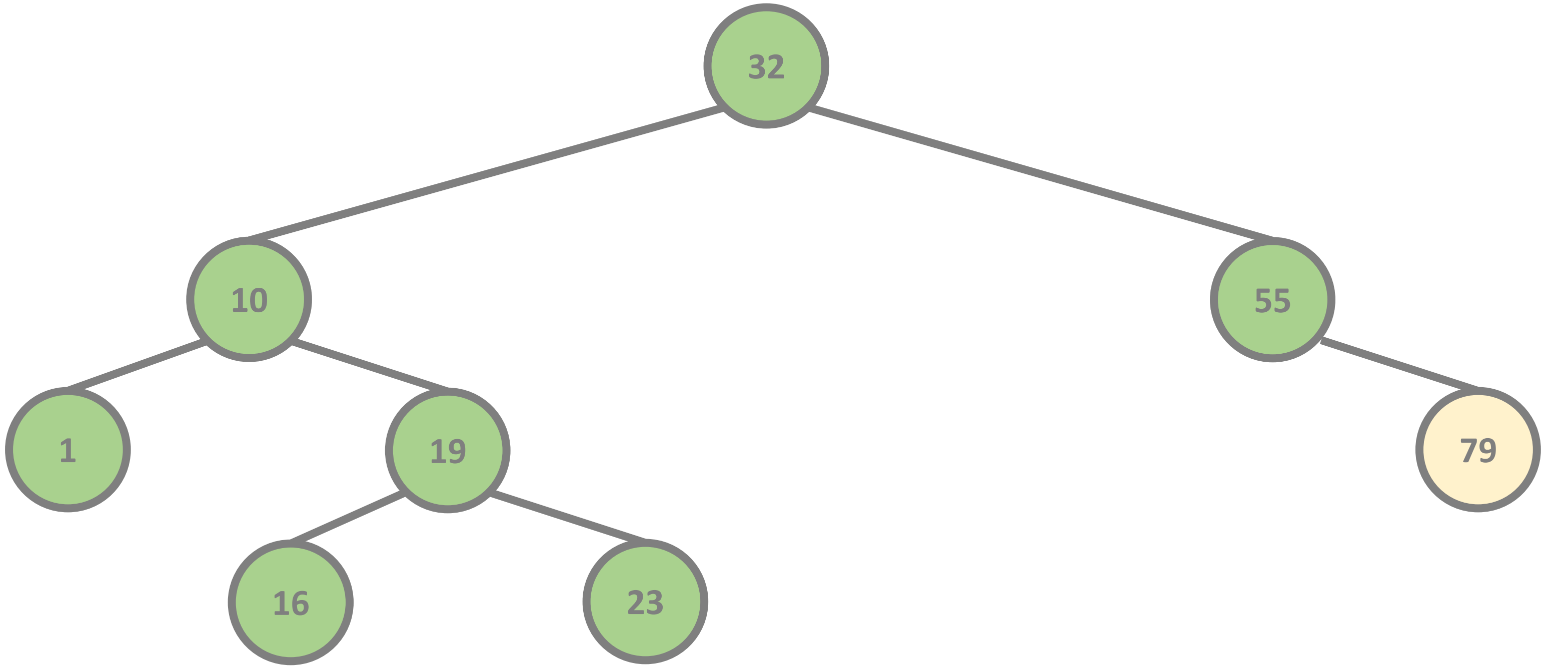
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

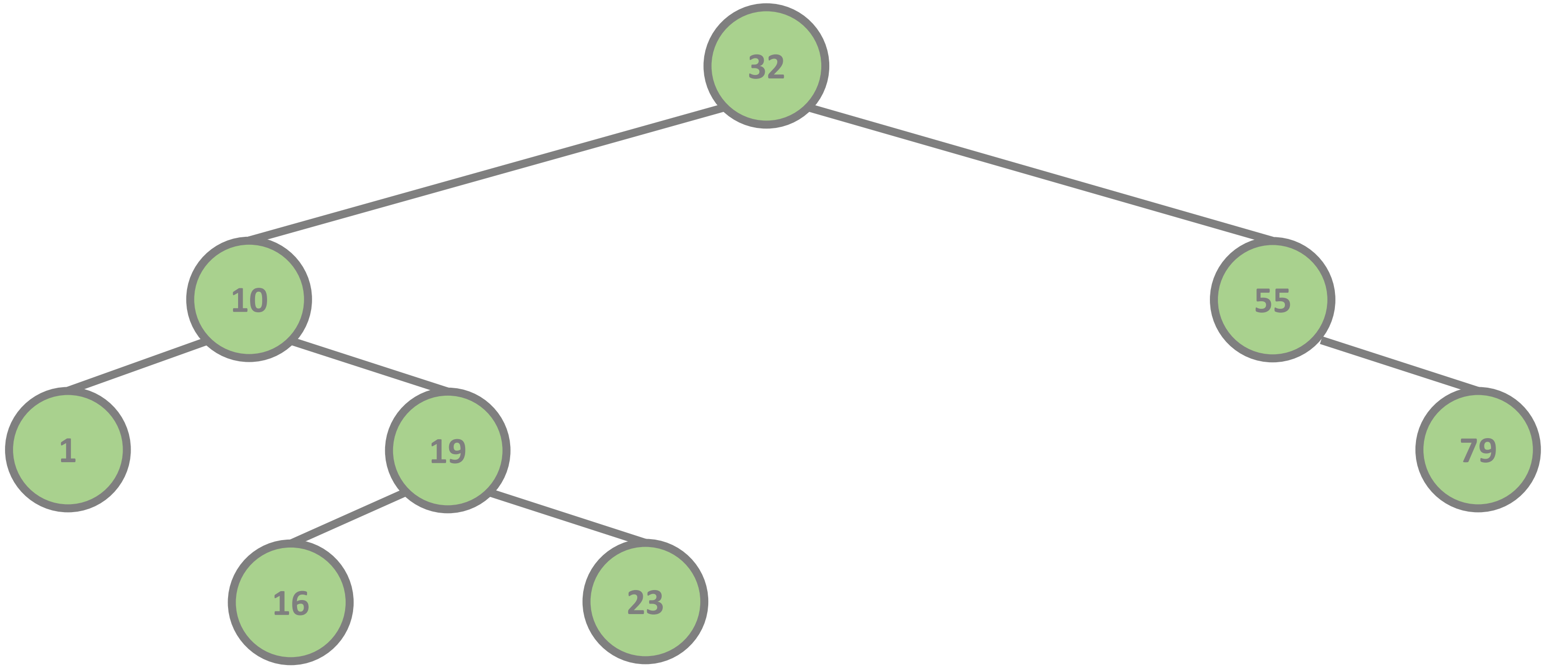
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

PRE-ORDER TRAVERSAL

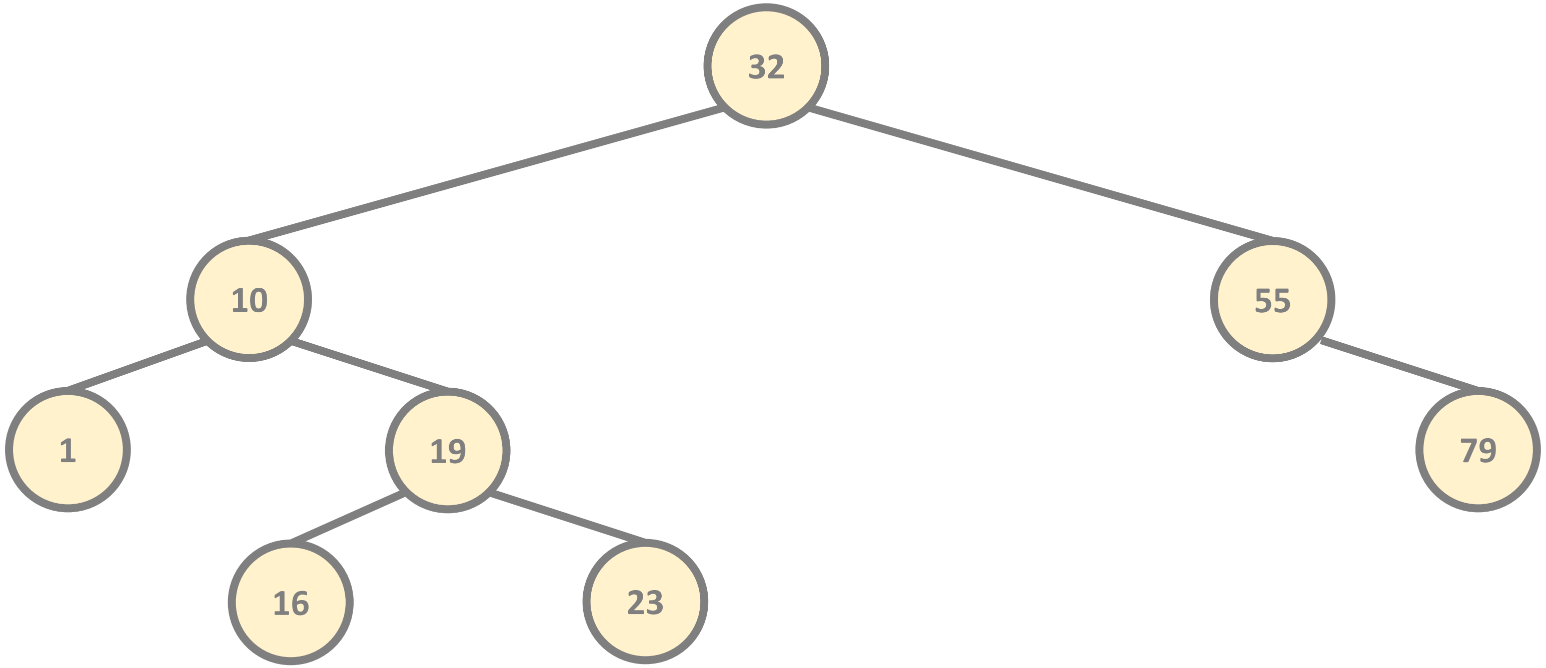
We visit the **root** node of the binary tree then the **left subtree** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

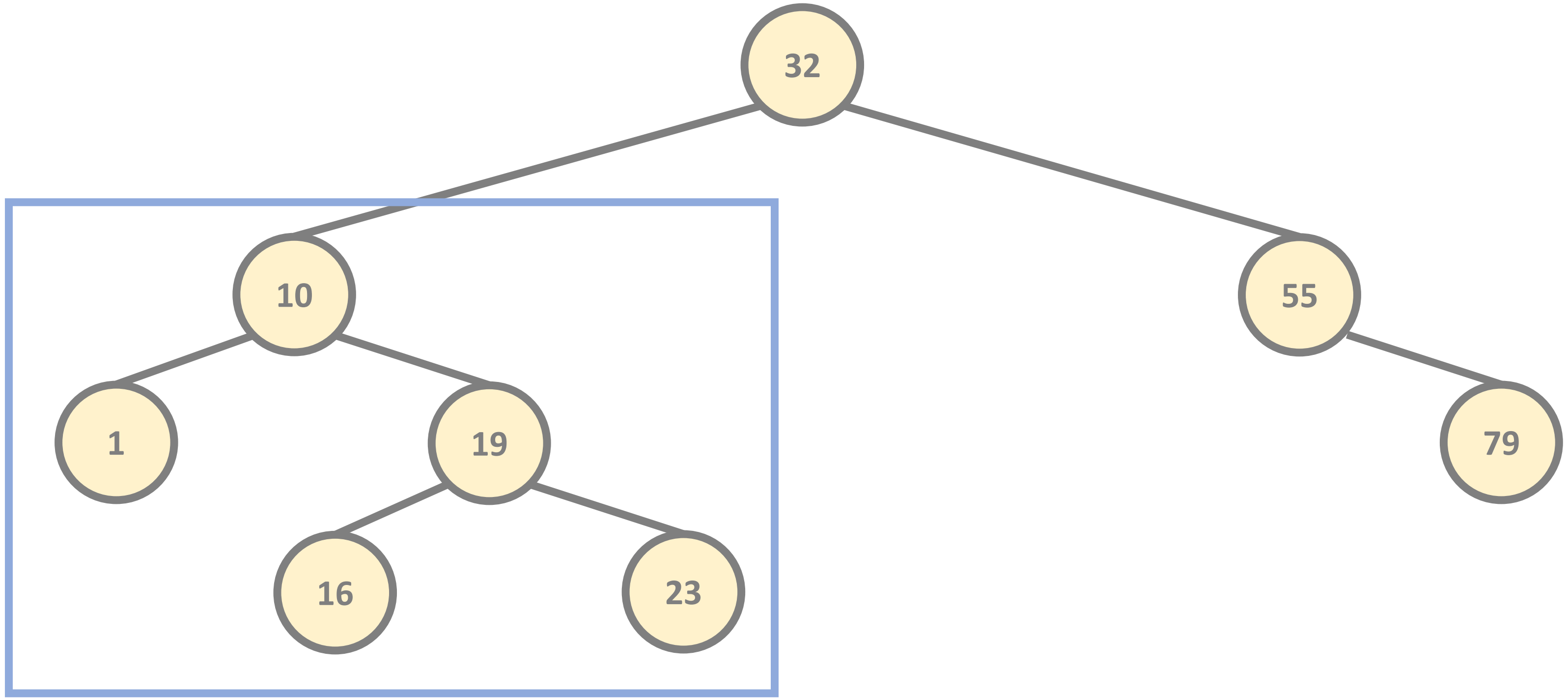
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

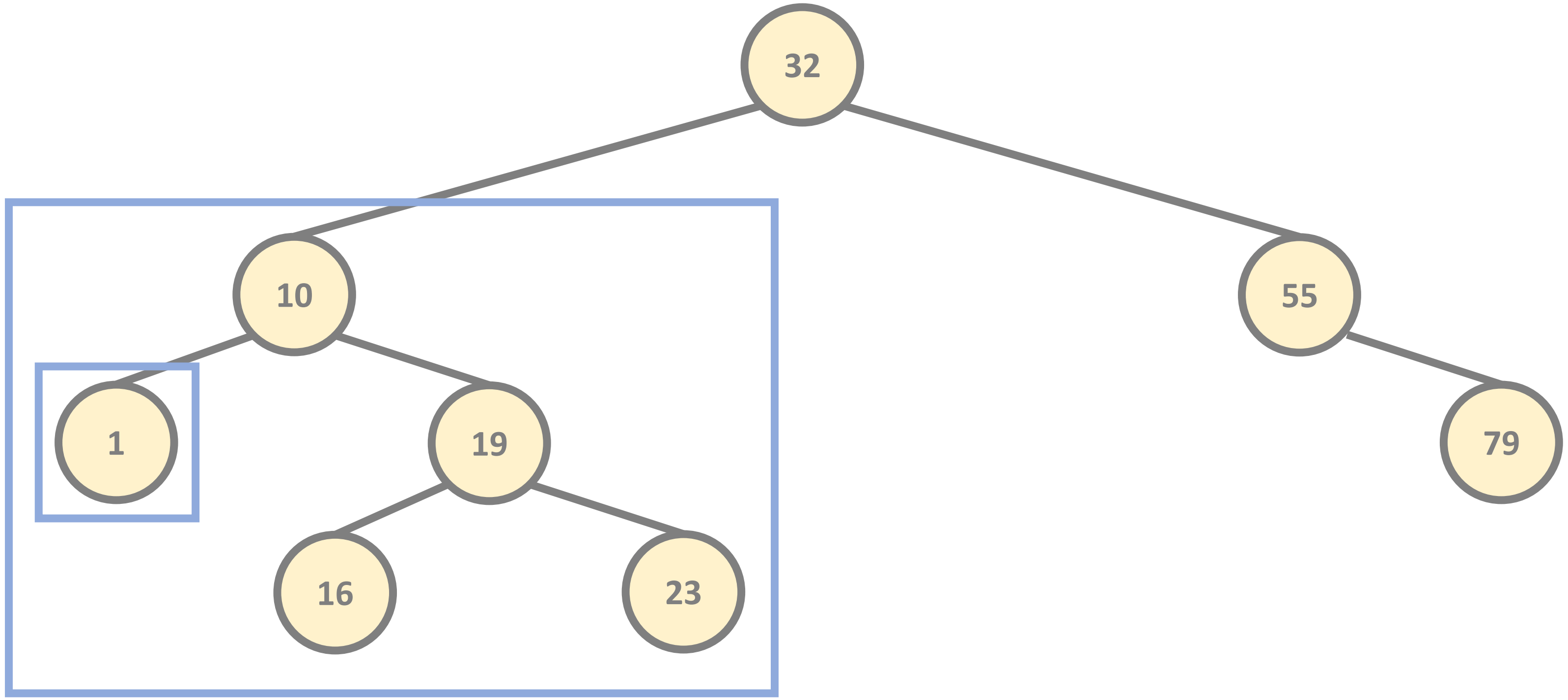
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

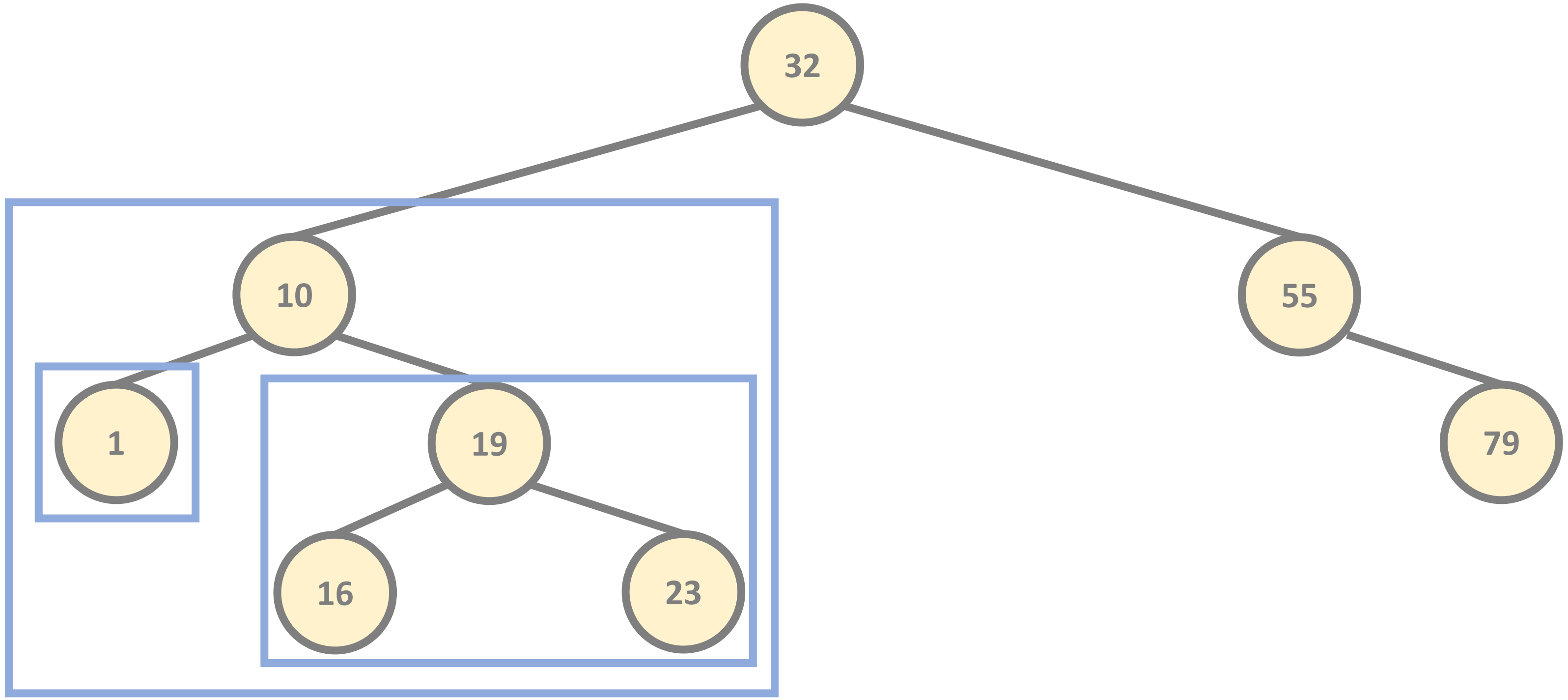
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

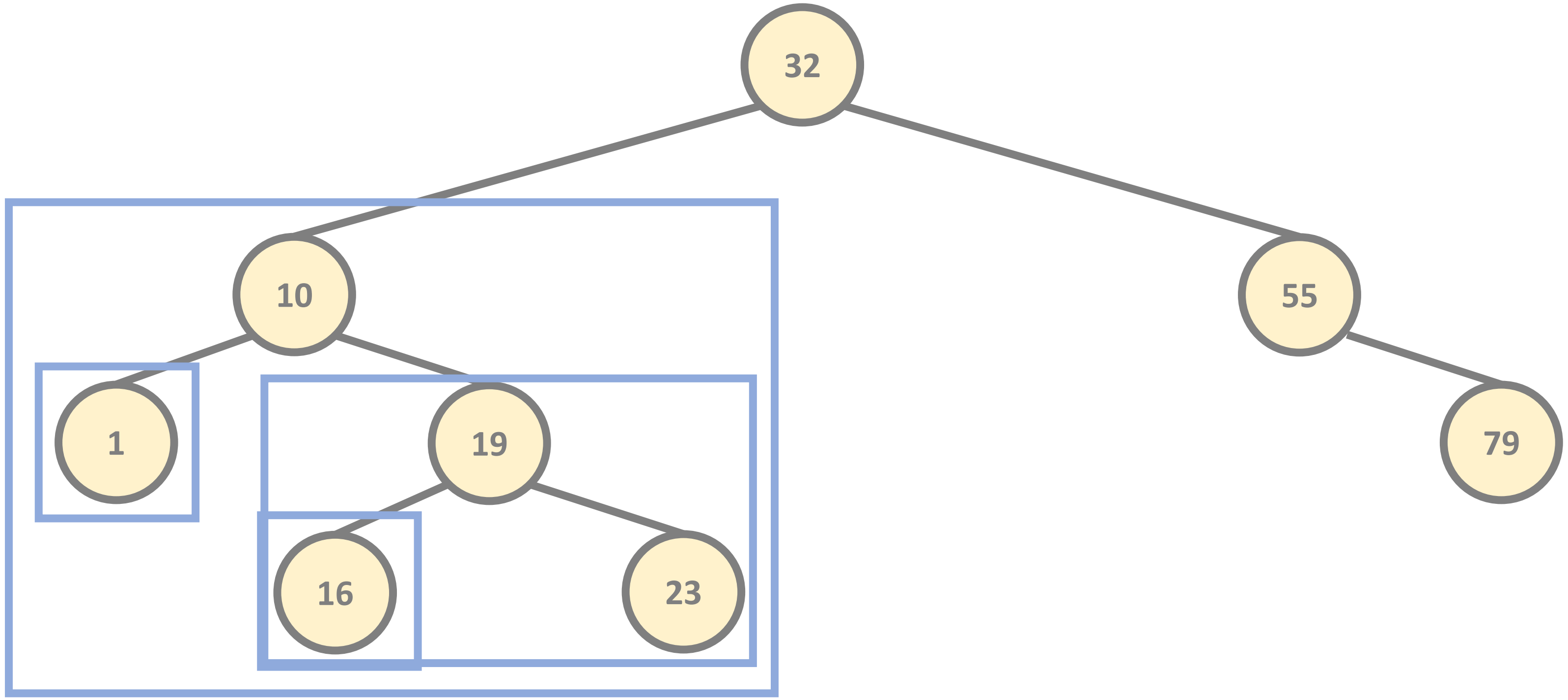
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

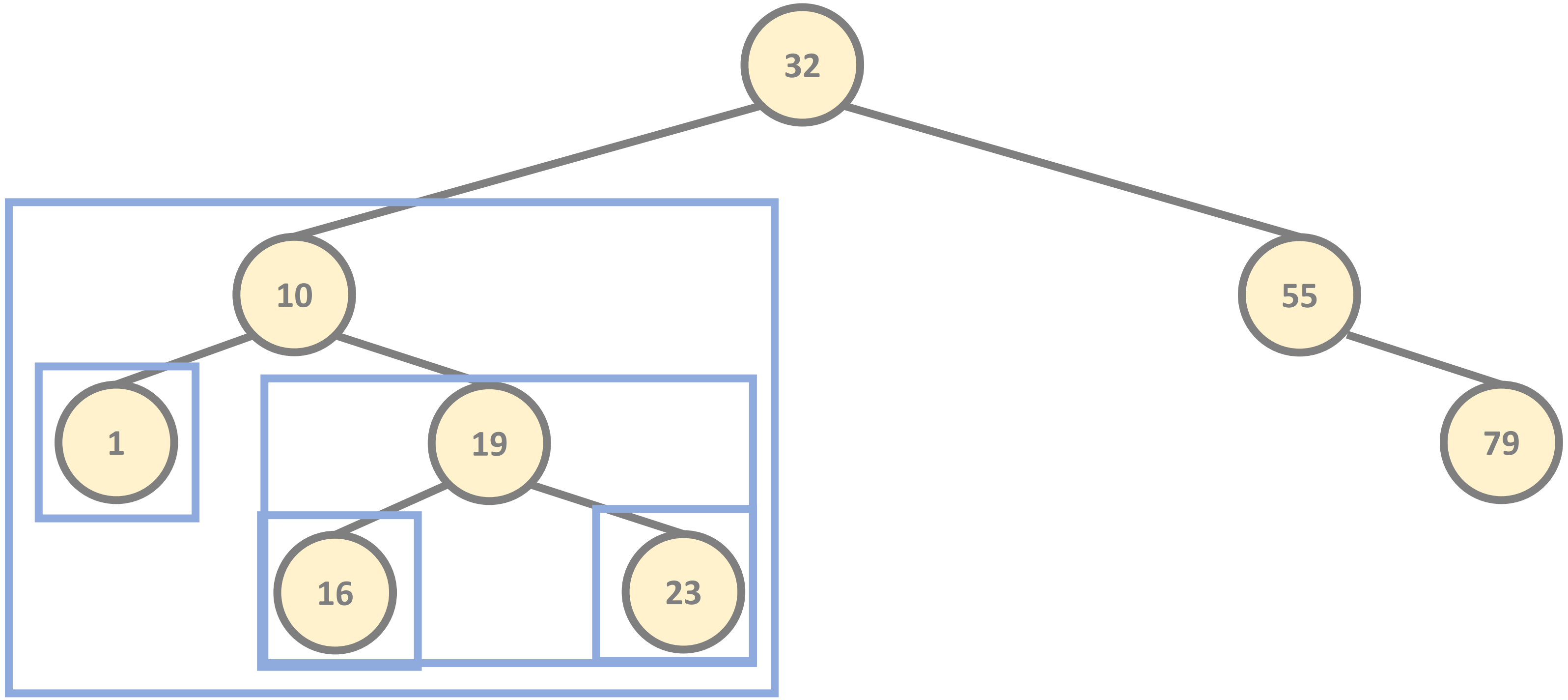
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

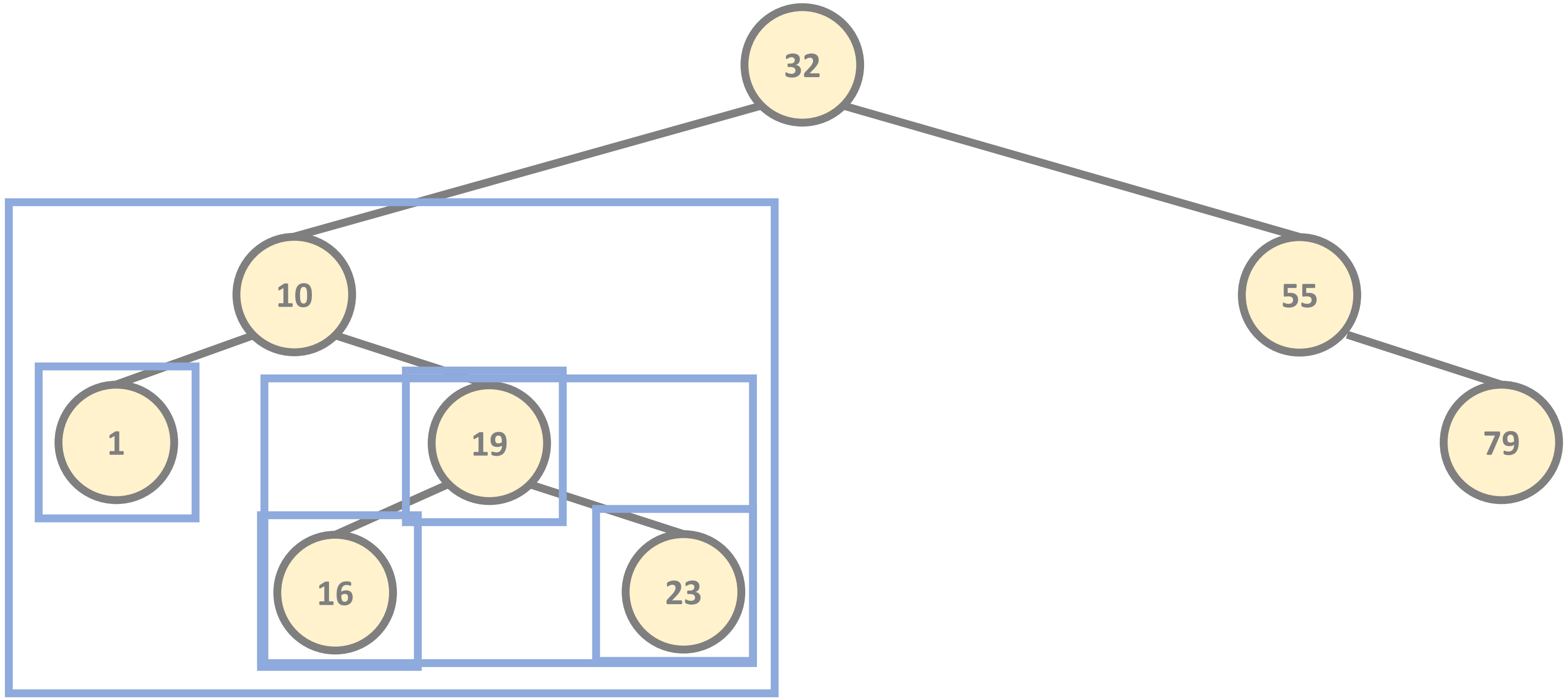
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

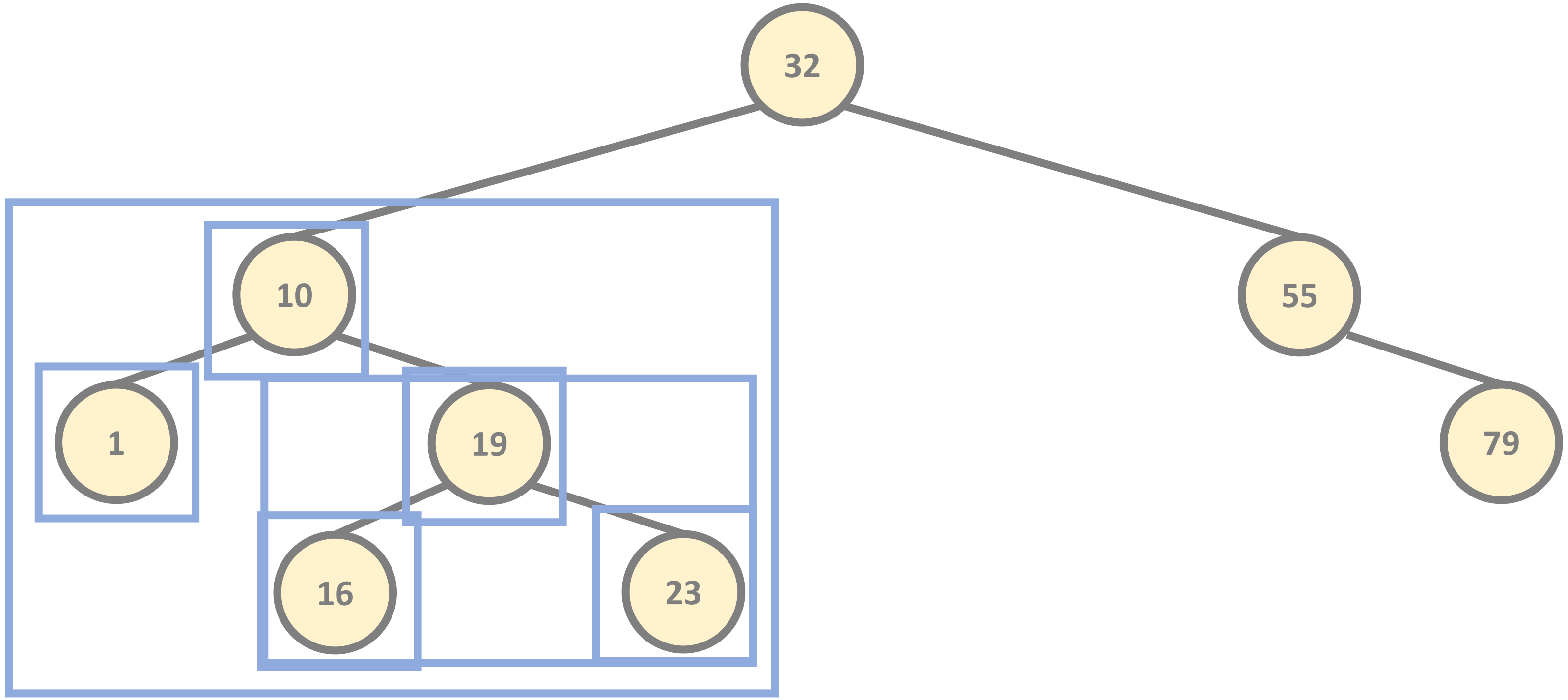
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

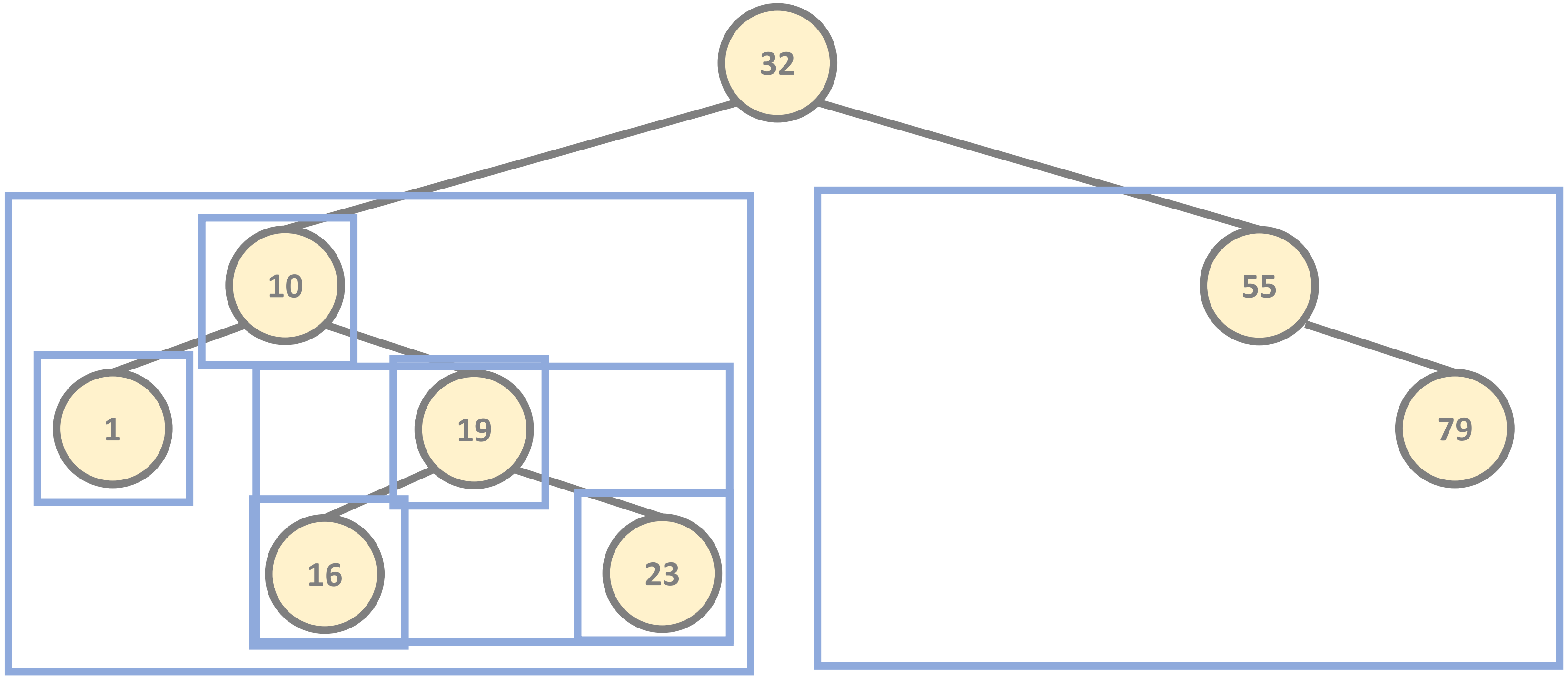
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

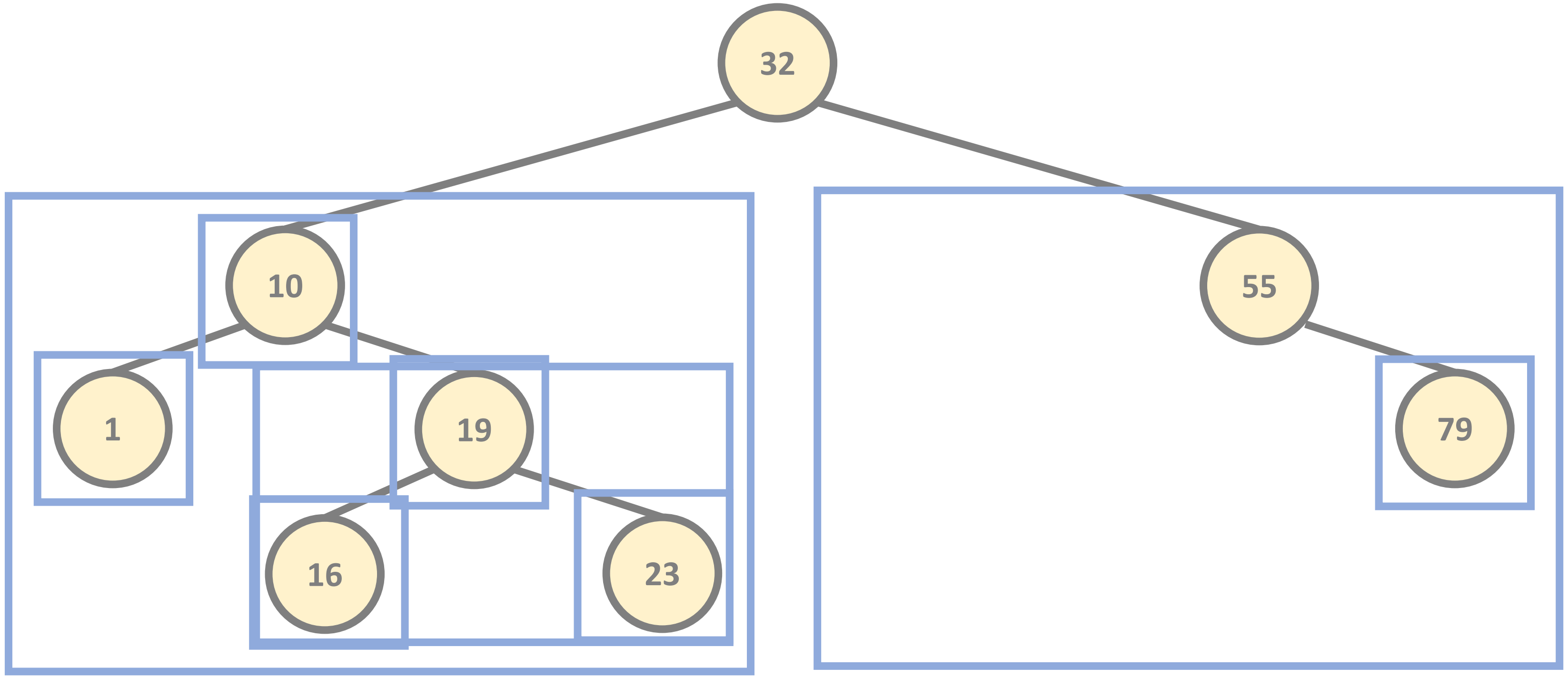
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

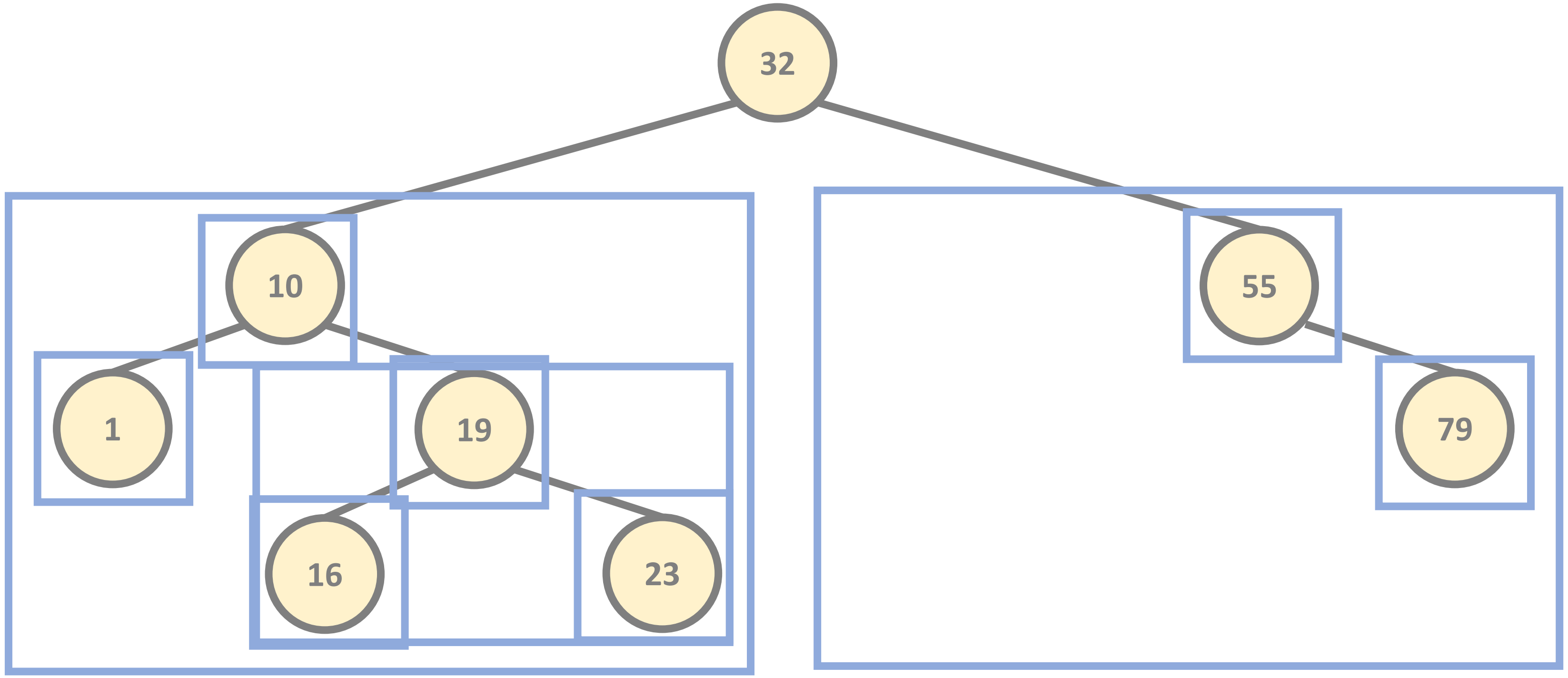
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

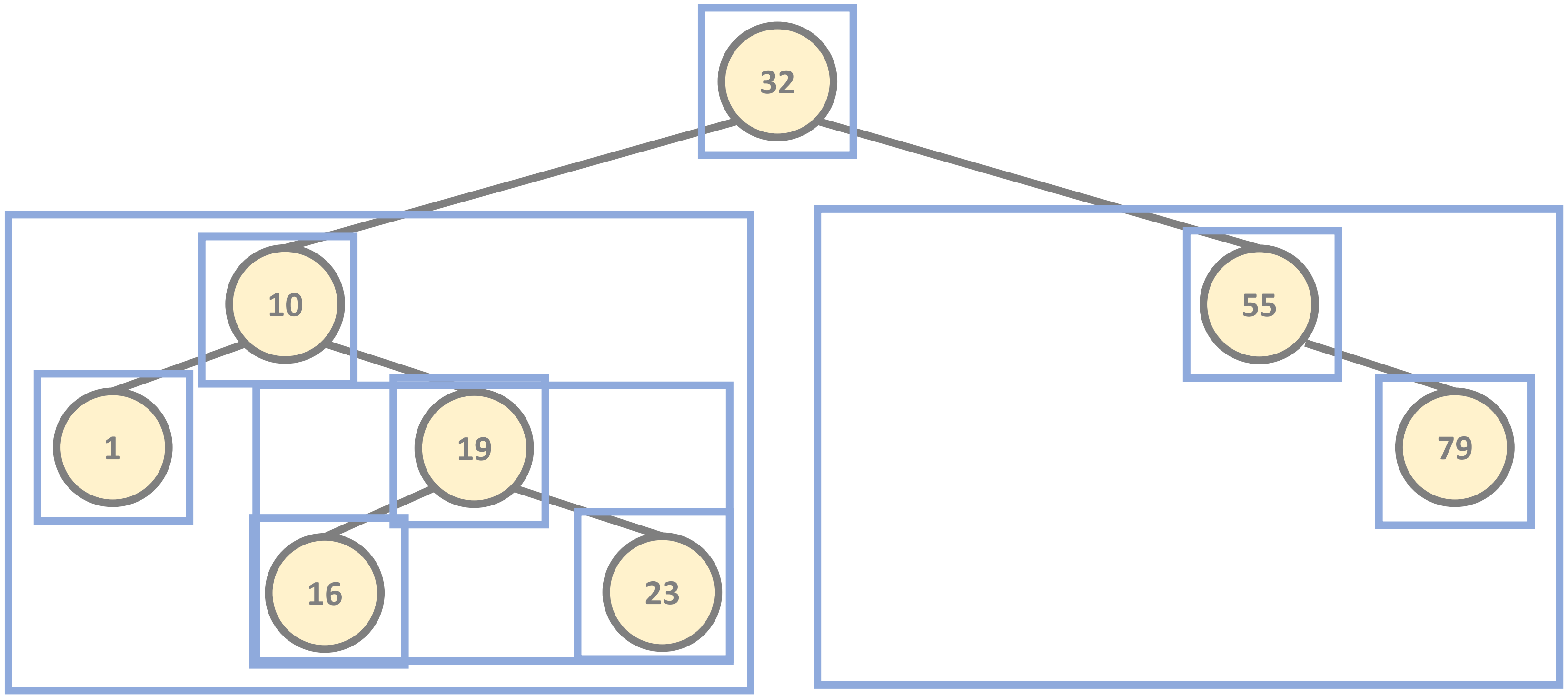
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

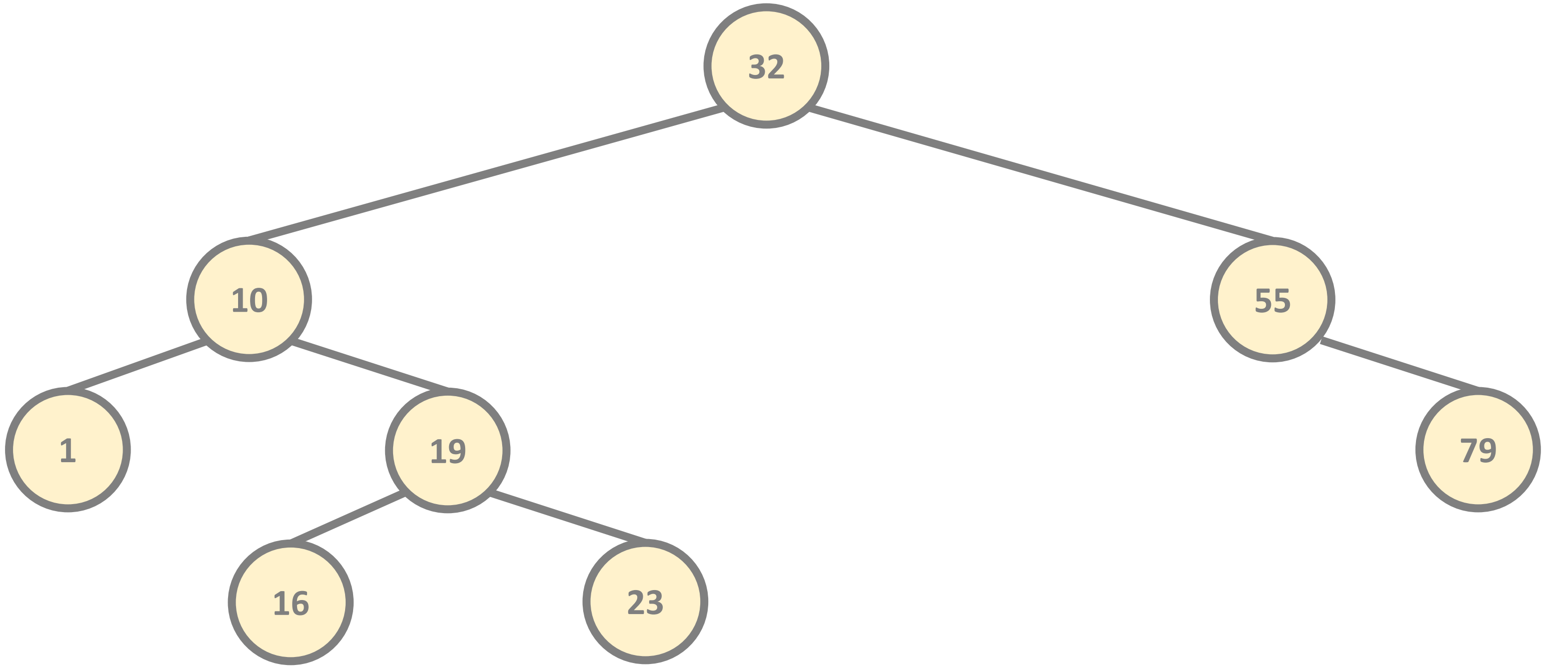
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

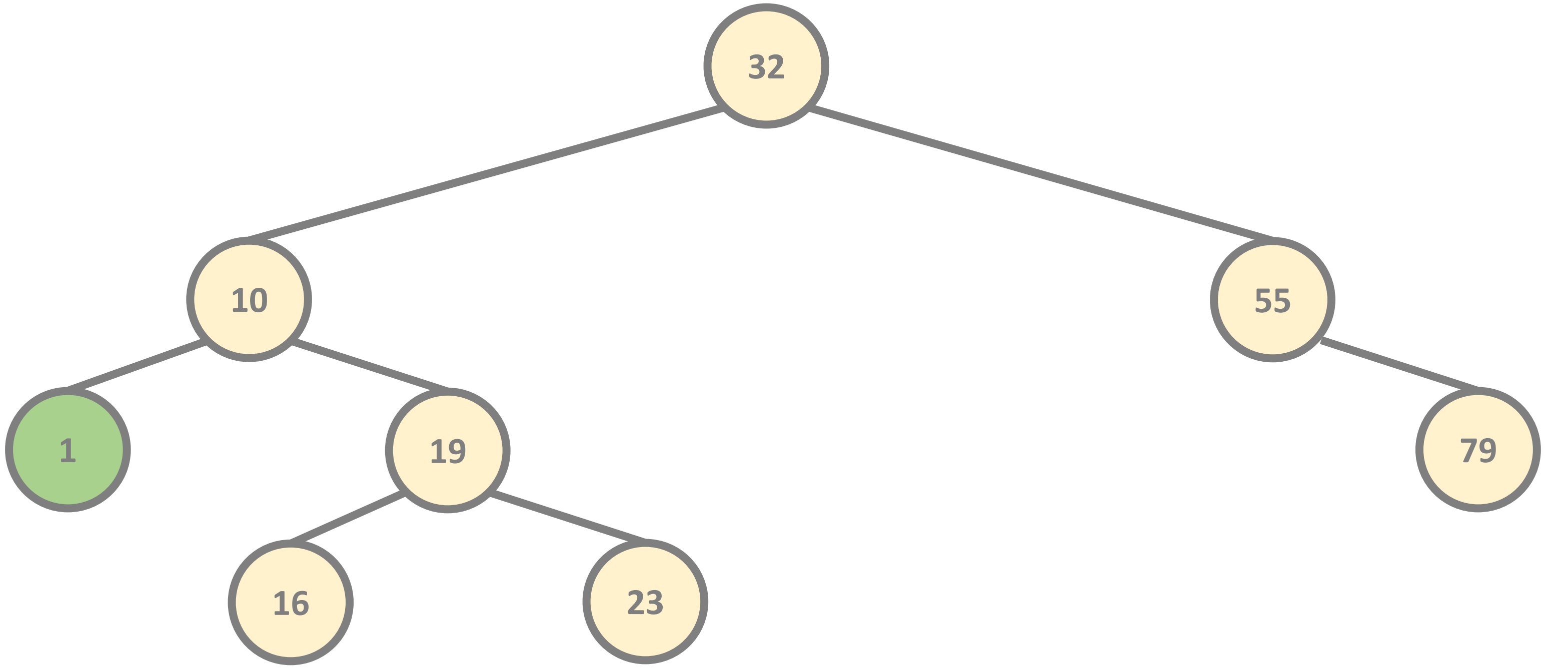
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

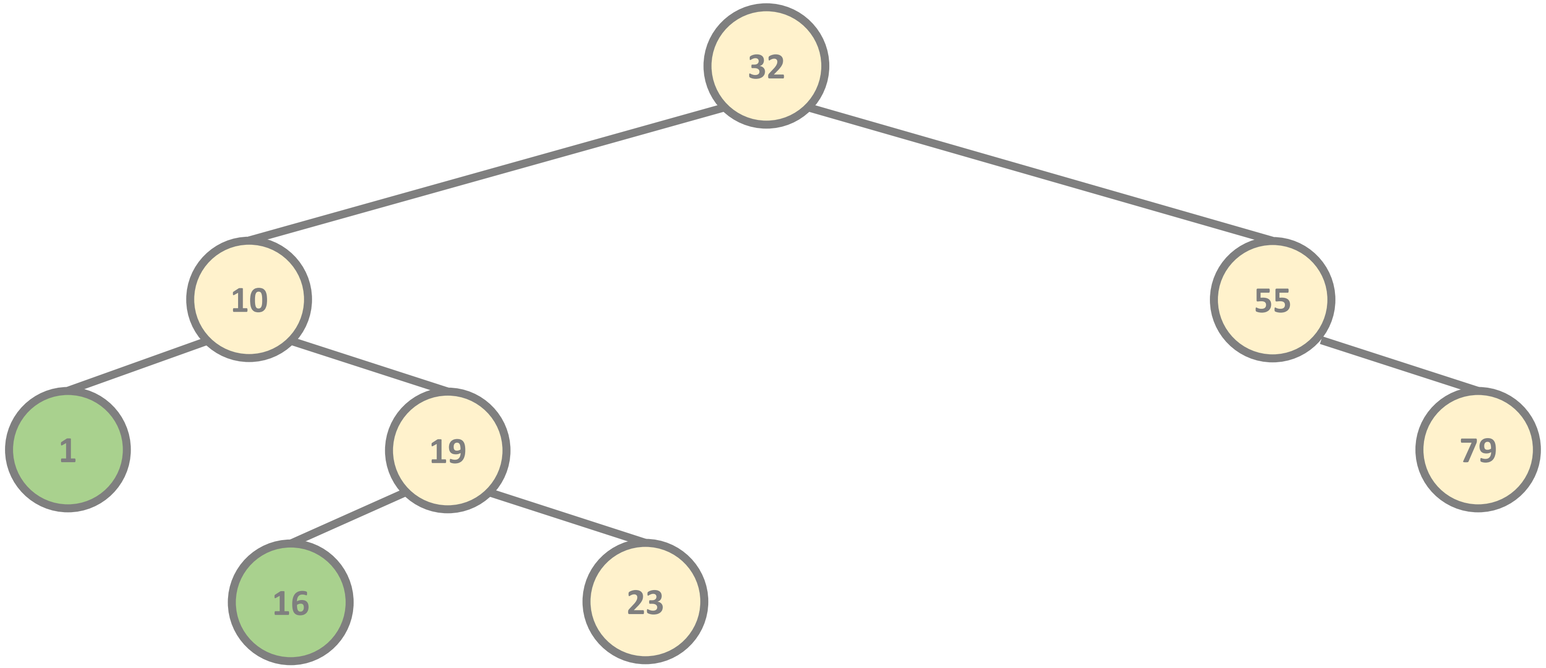
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

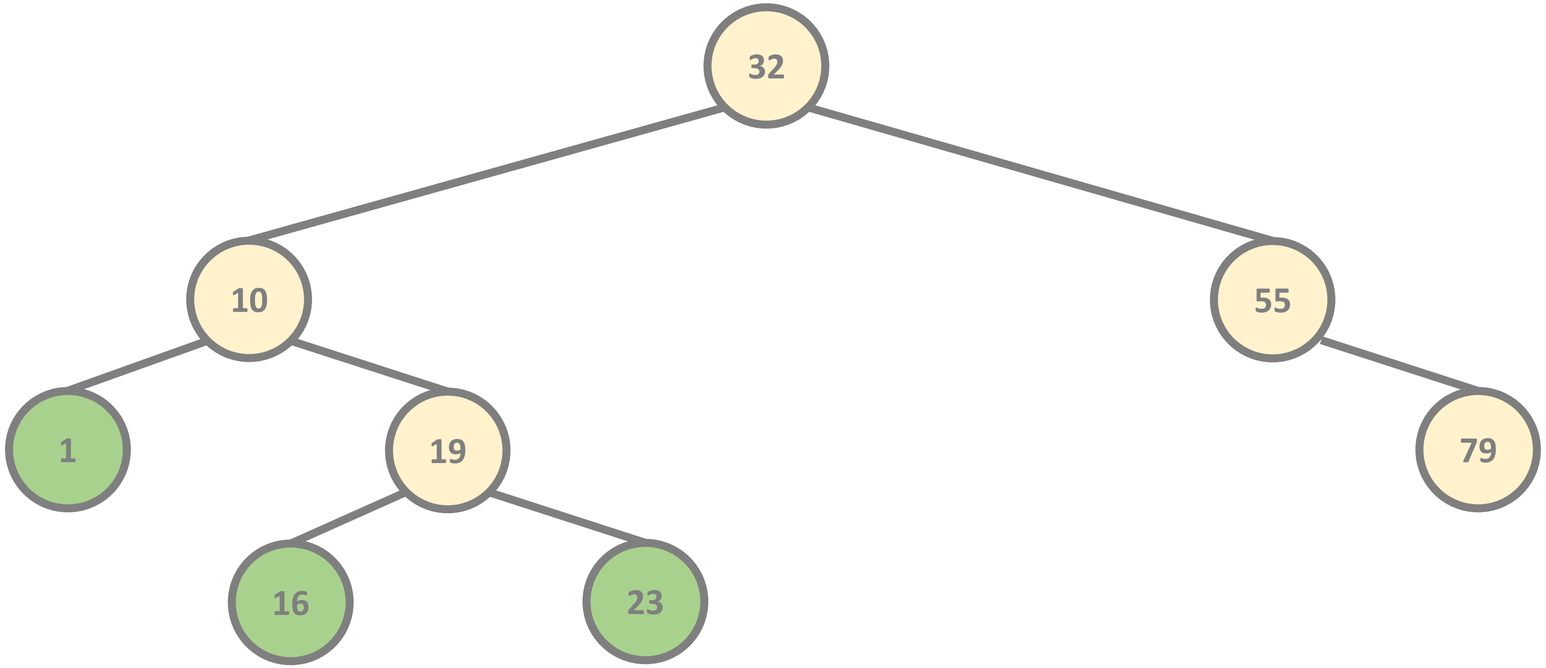
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

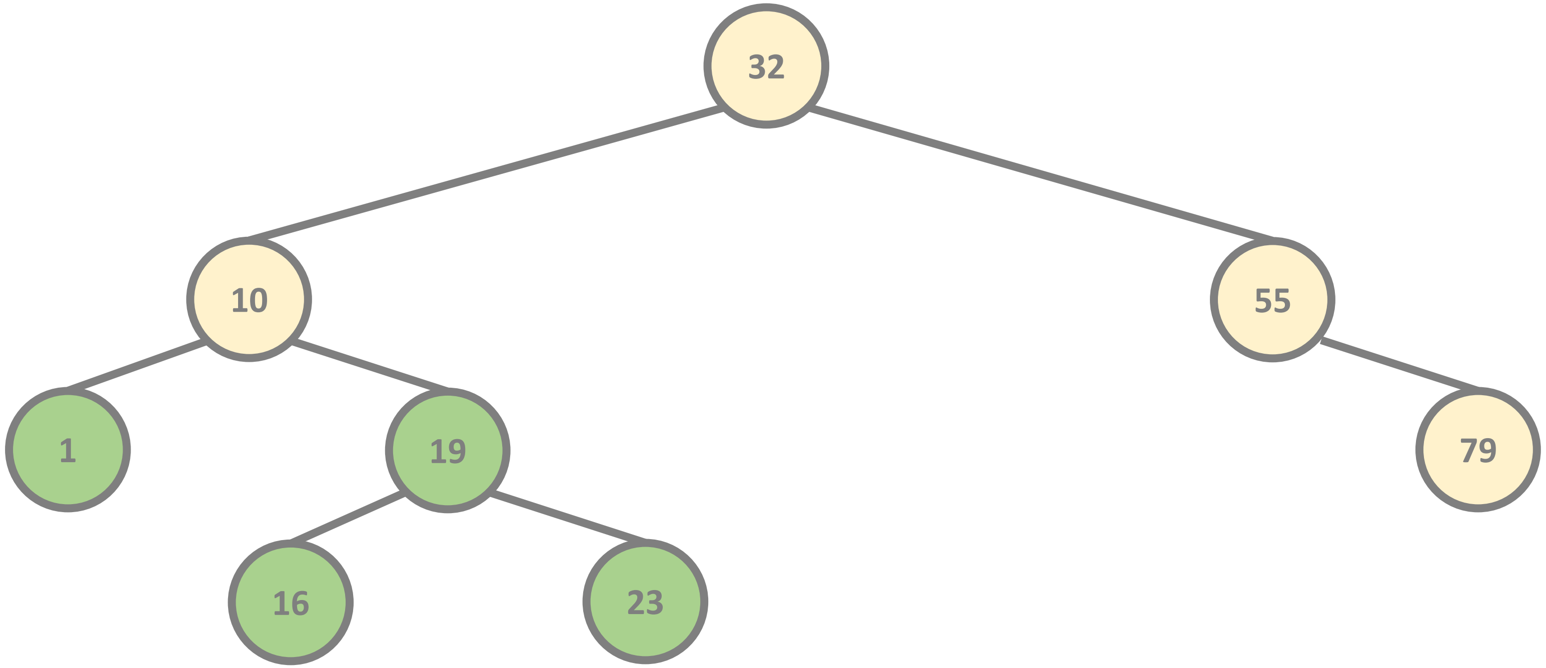
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

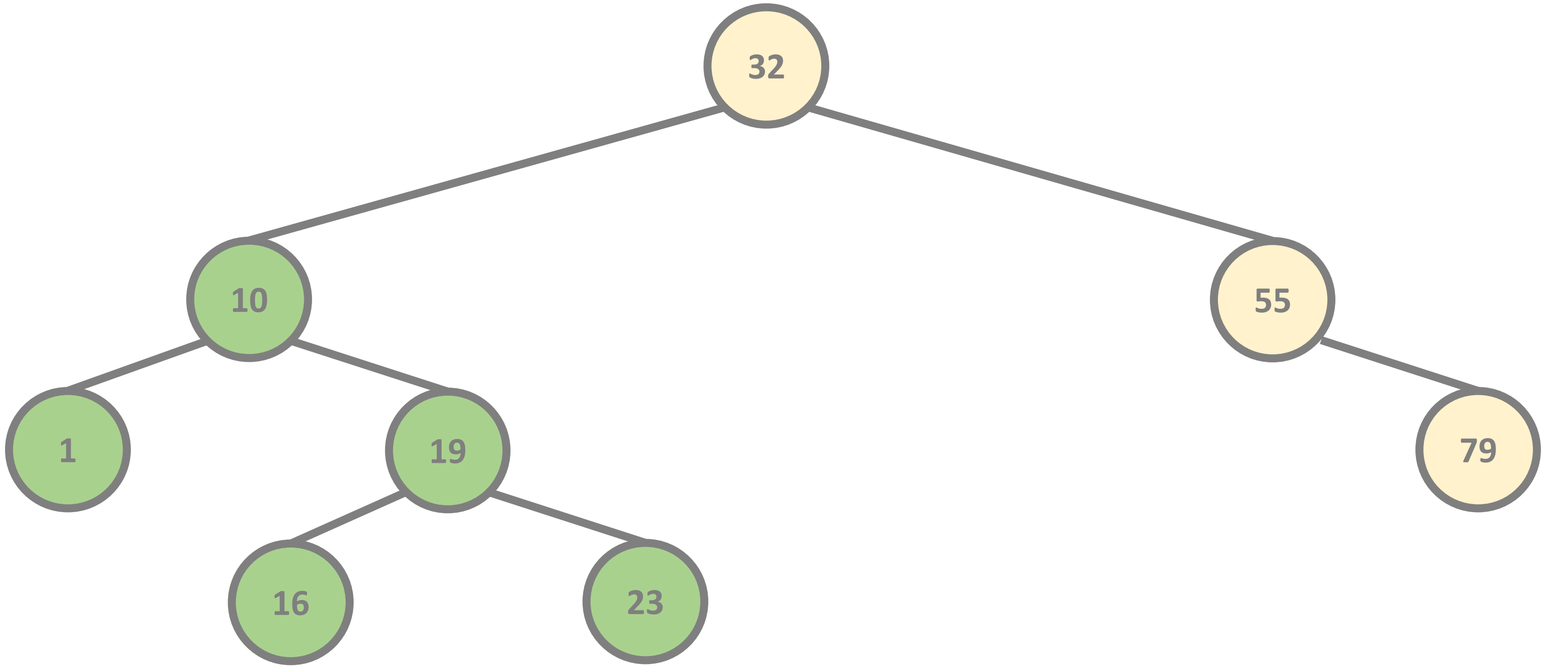
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

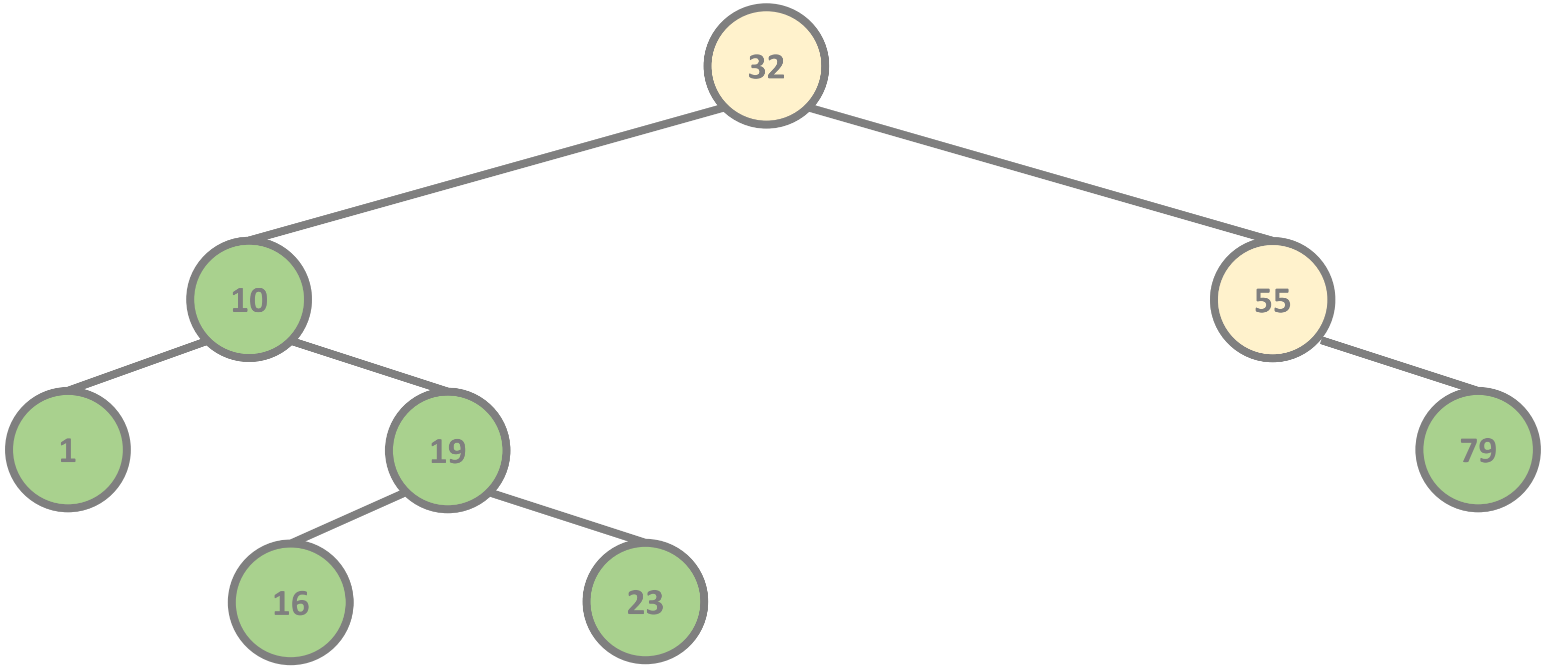
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

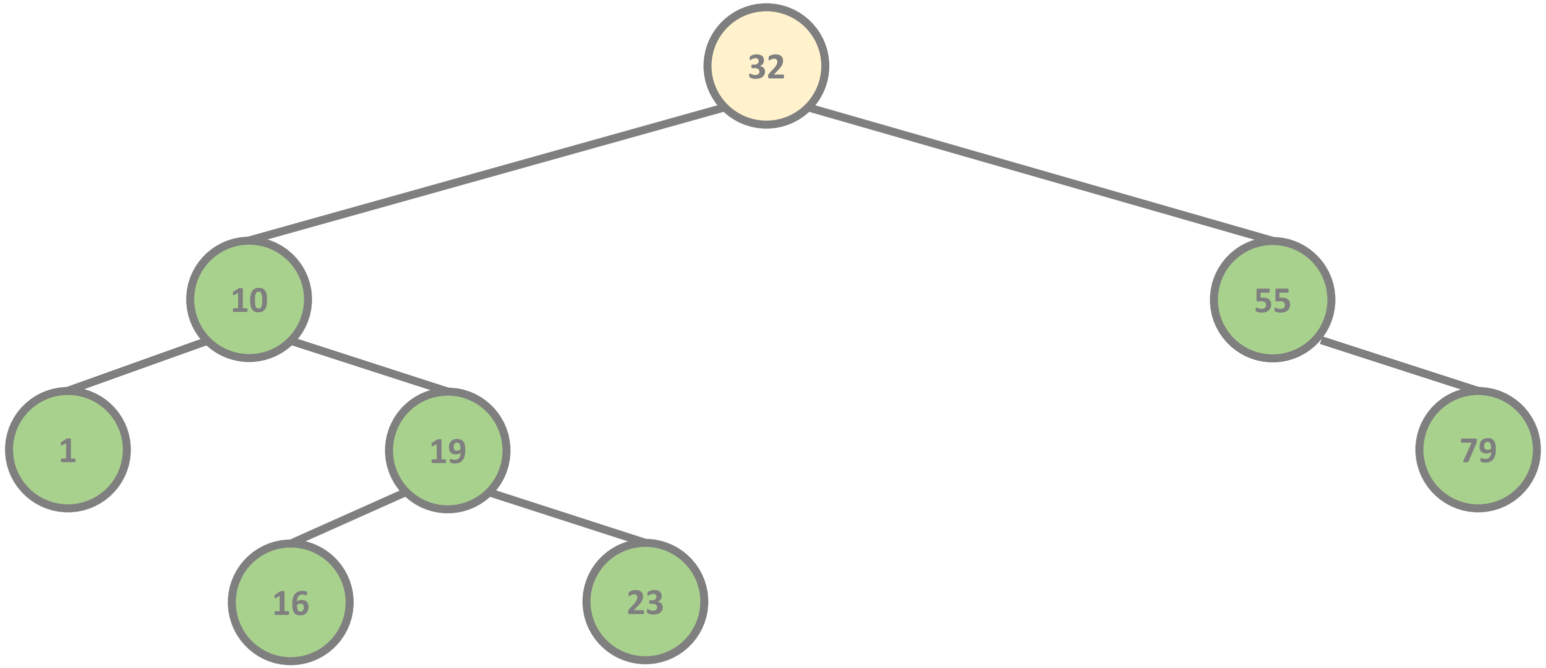
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

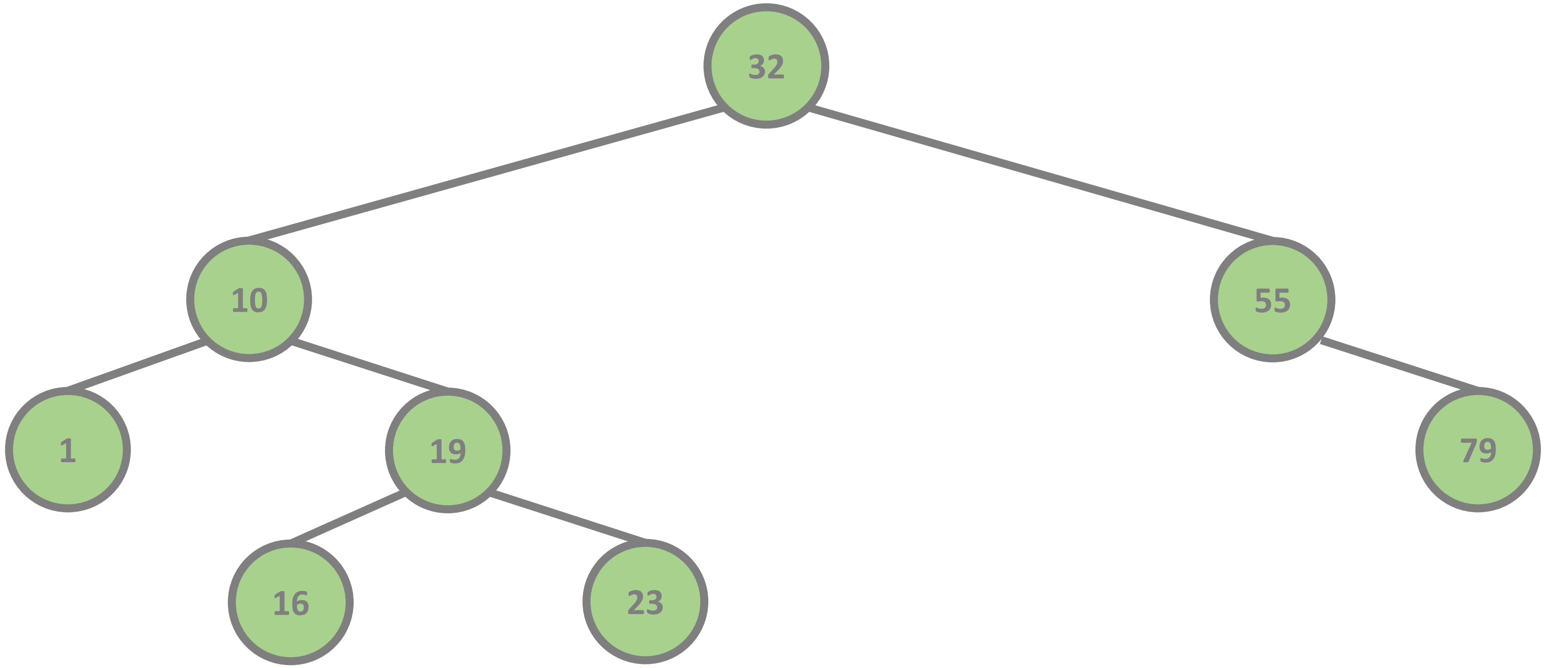
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

POST-ORDER TRAVERSAL

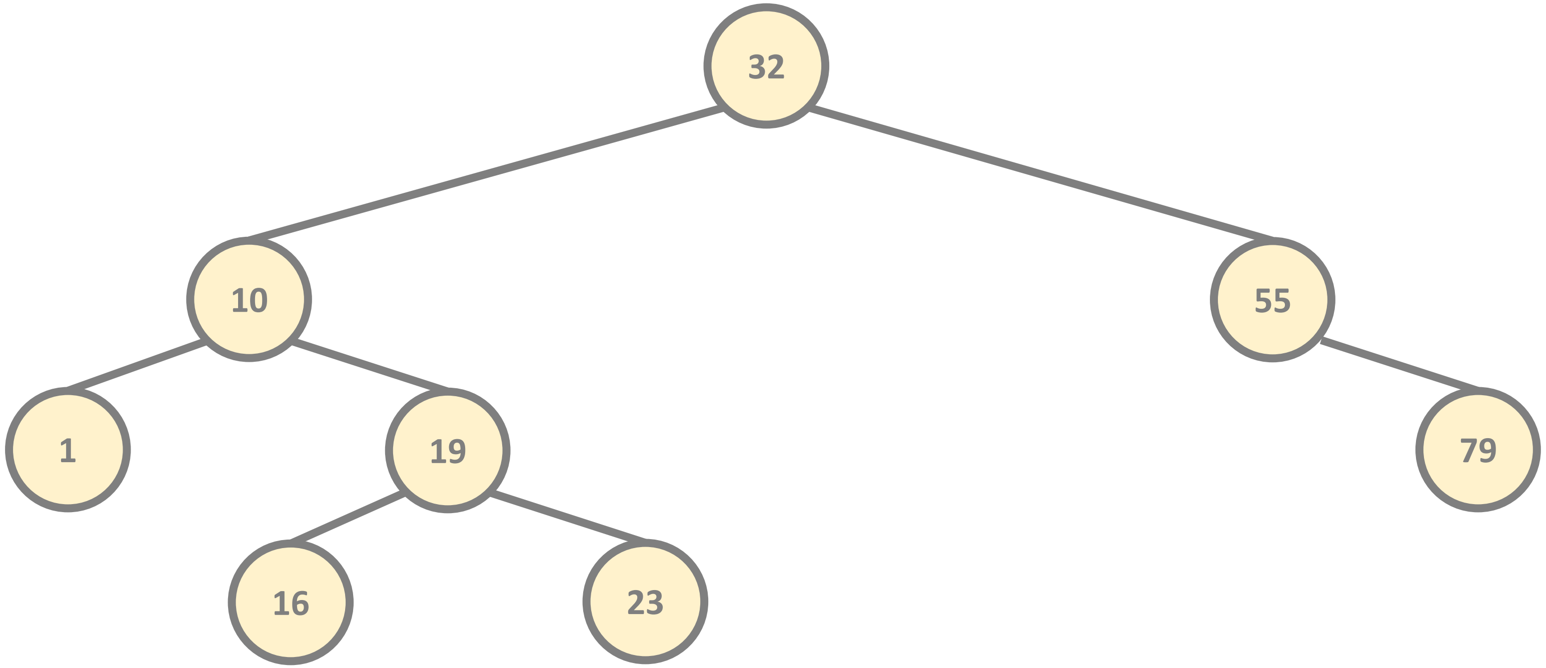
We visit the **left subtree** of the binary tree then the **right subtree** and finally the **root node** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

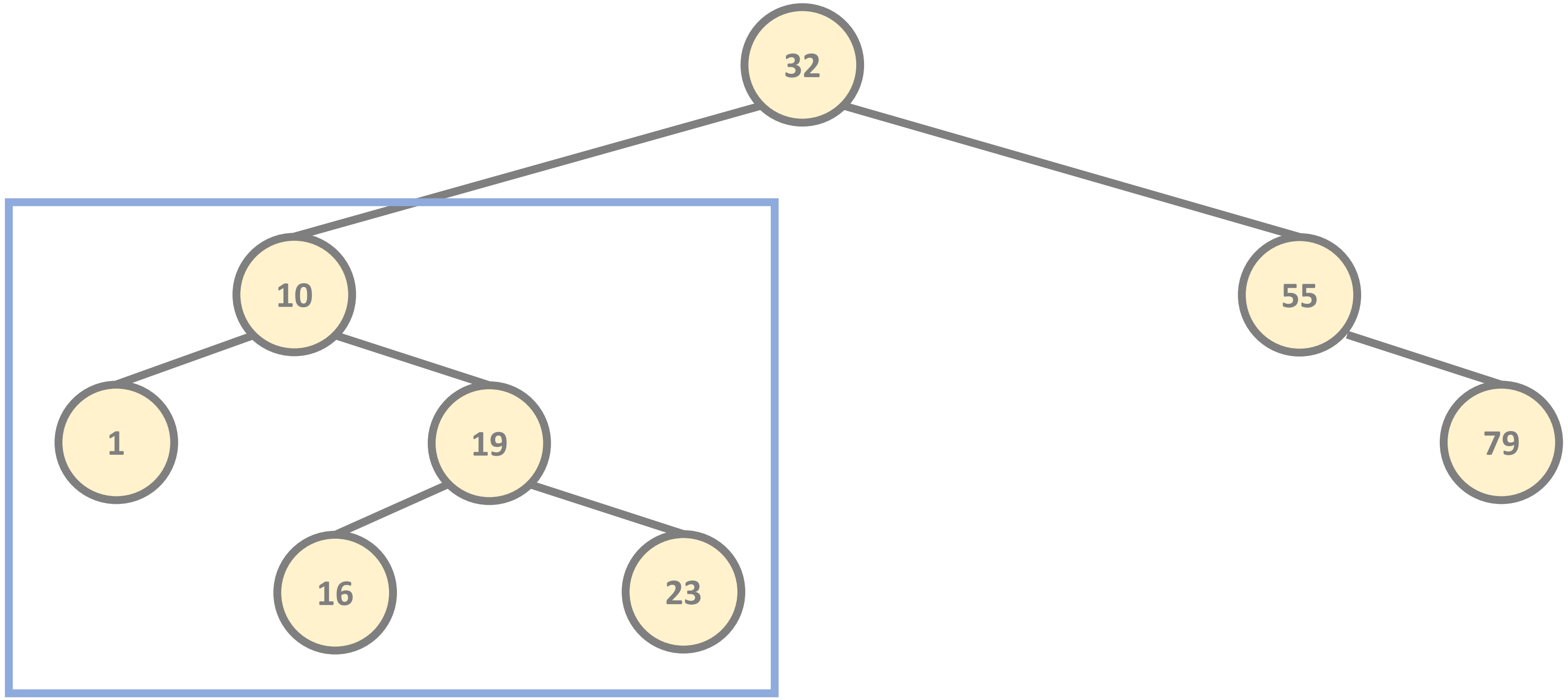
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

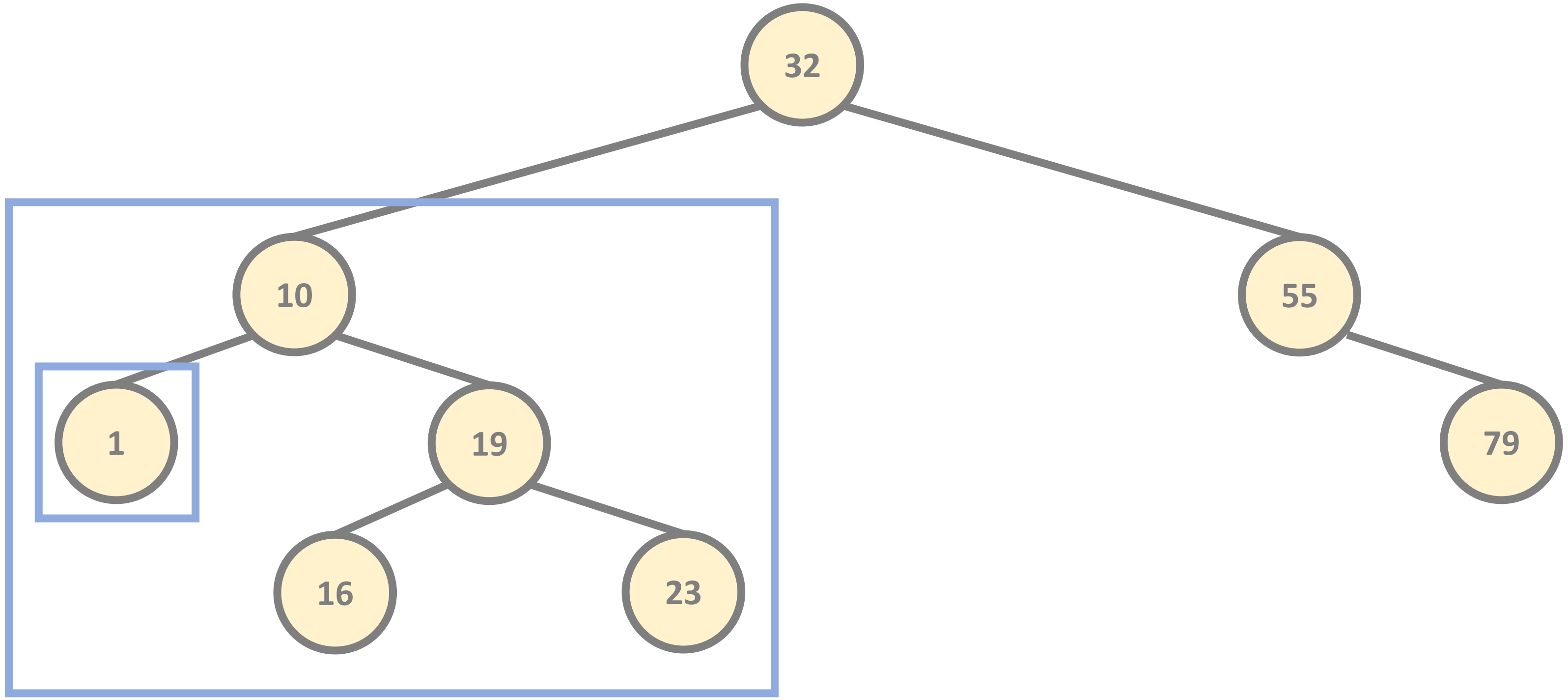
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

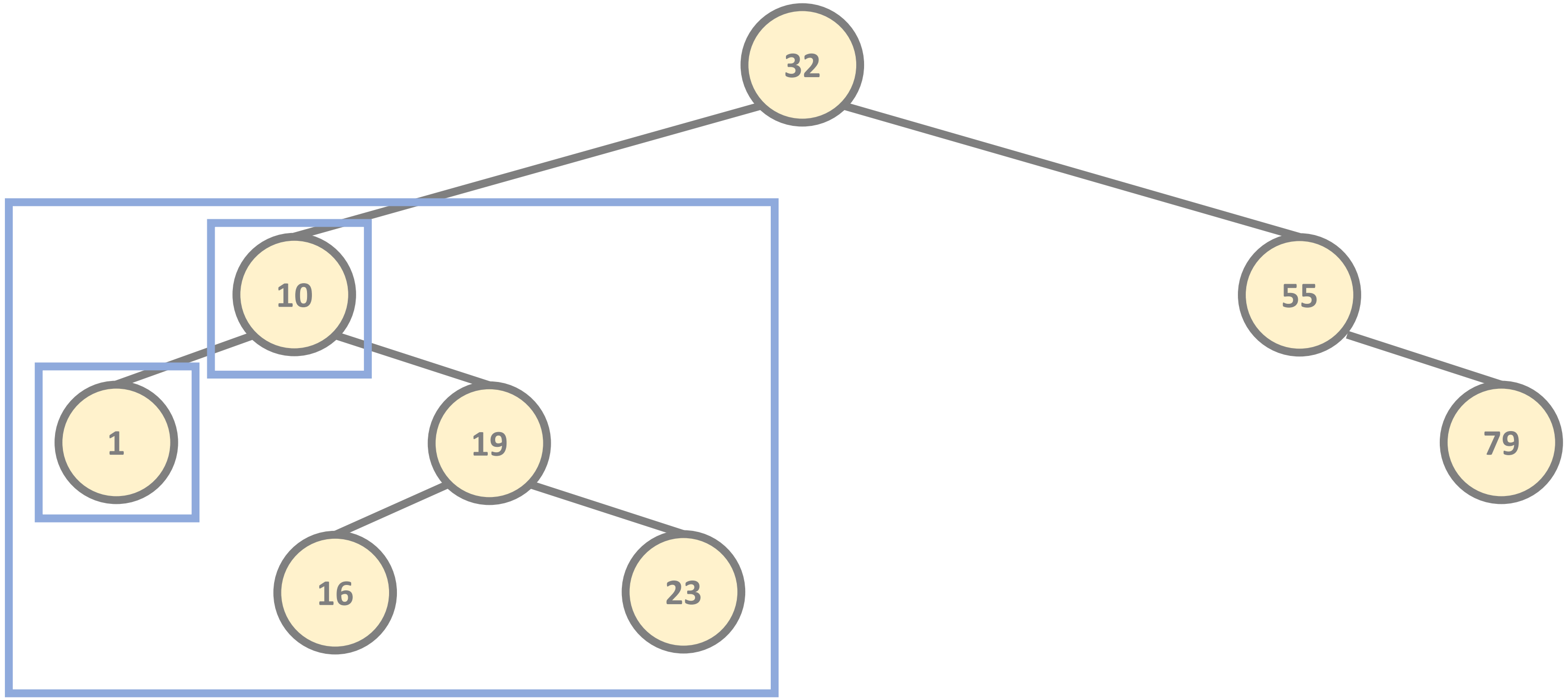
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

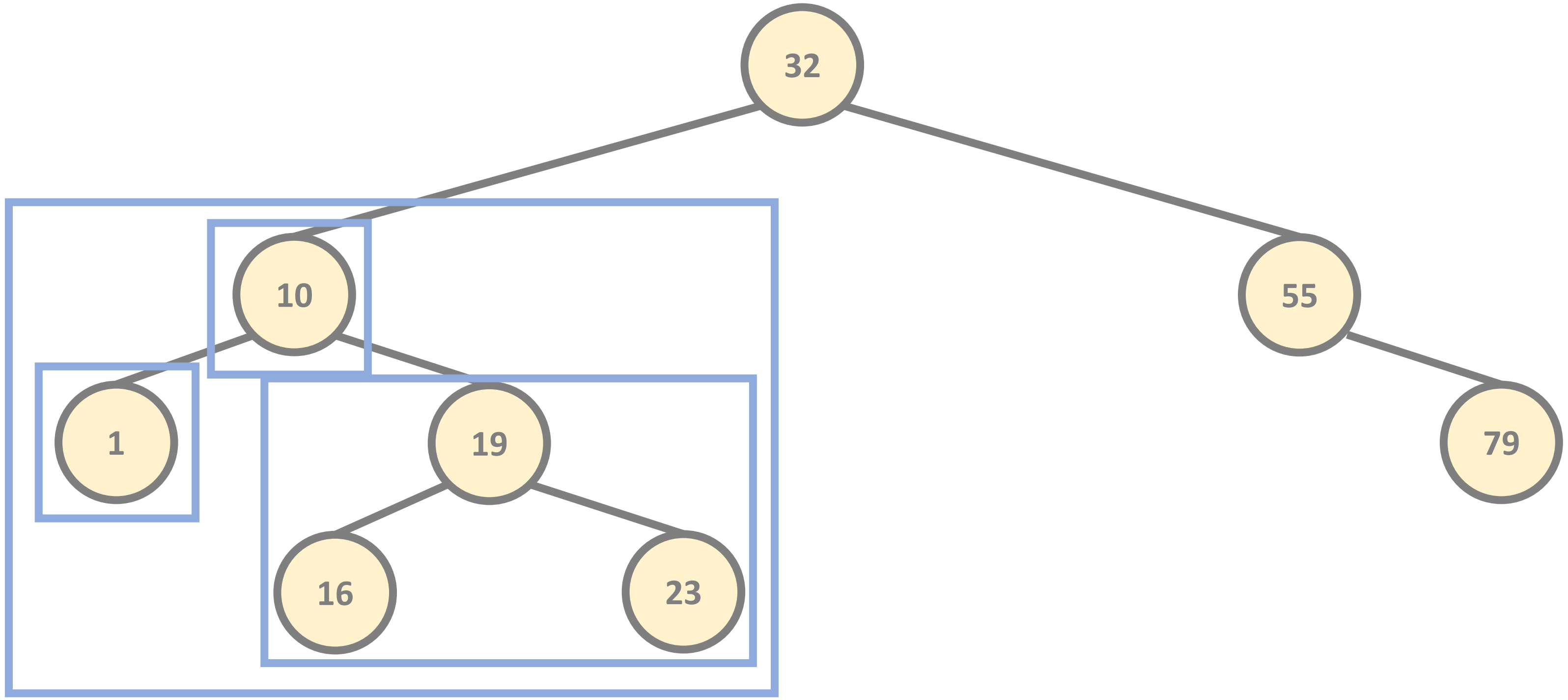
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

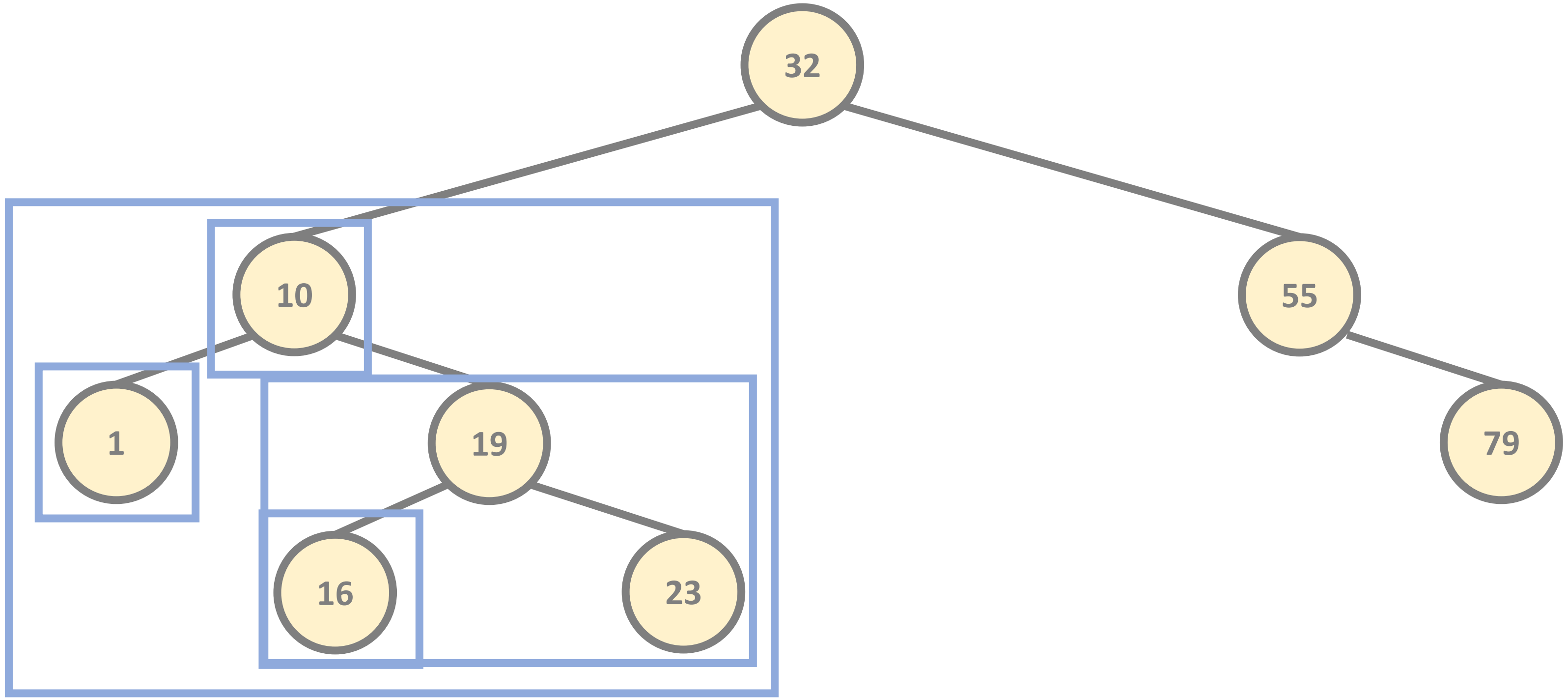
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

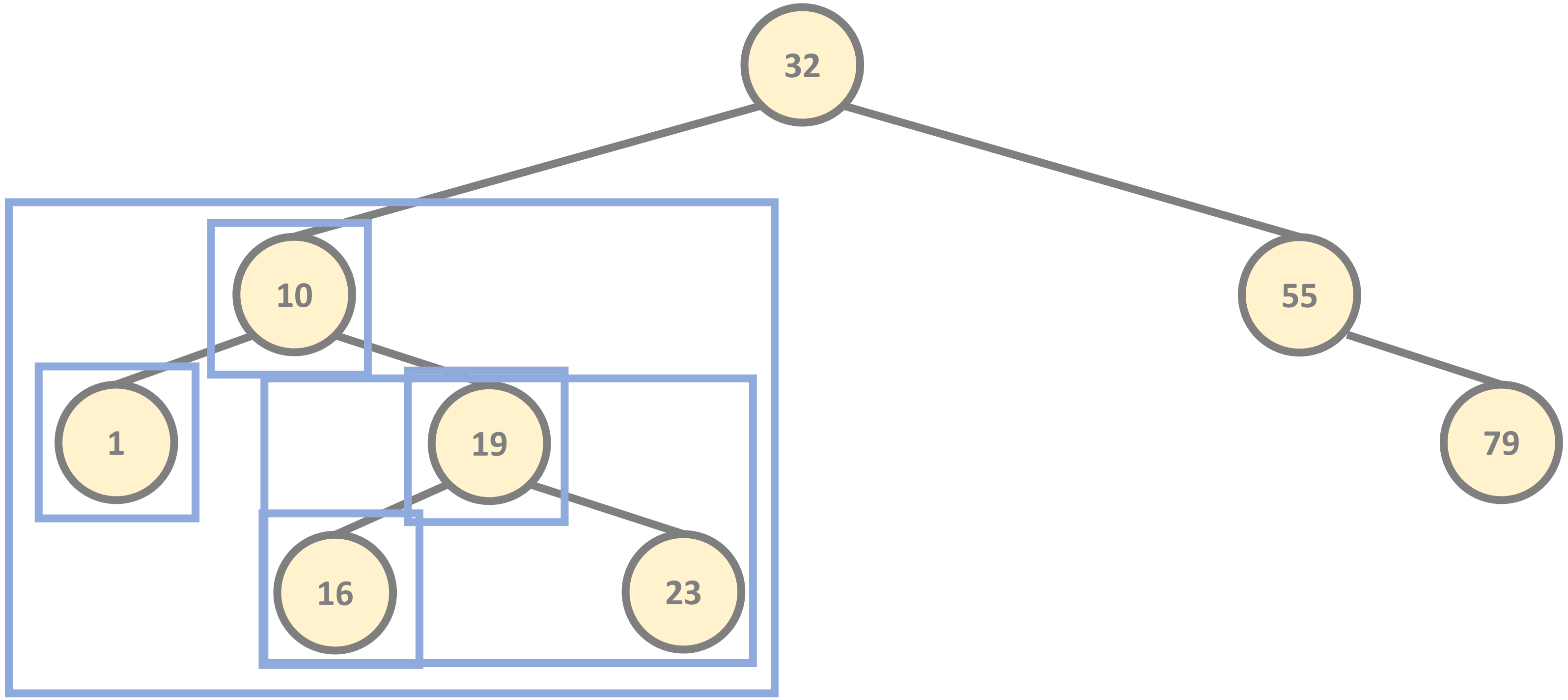
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

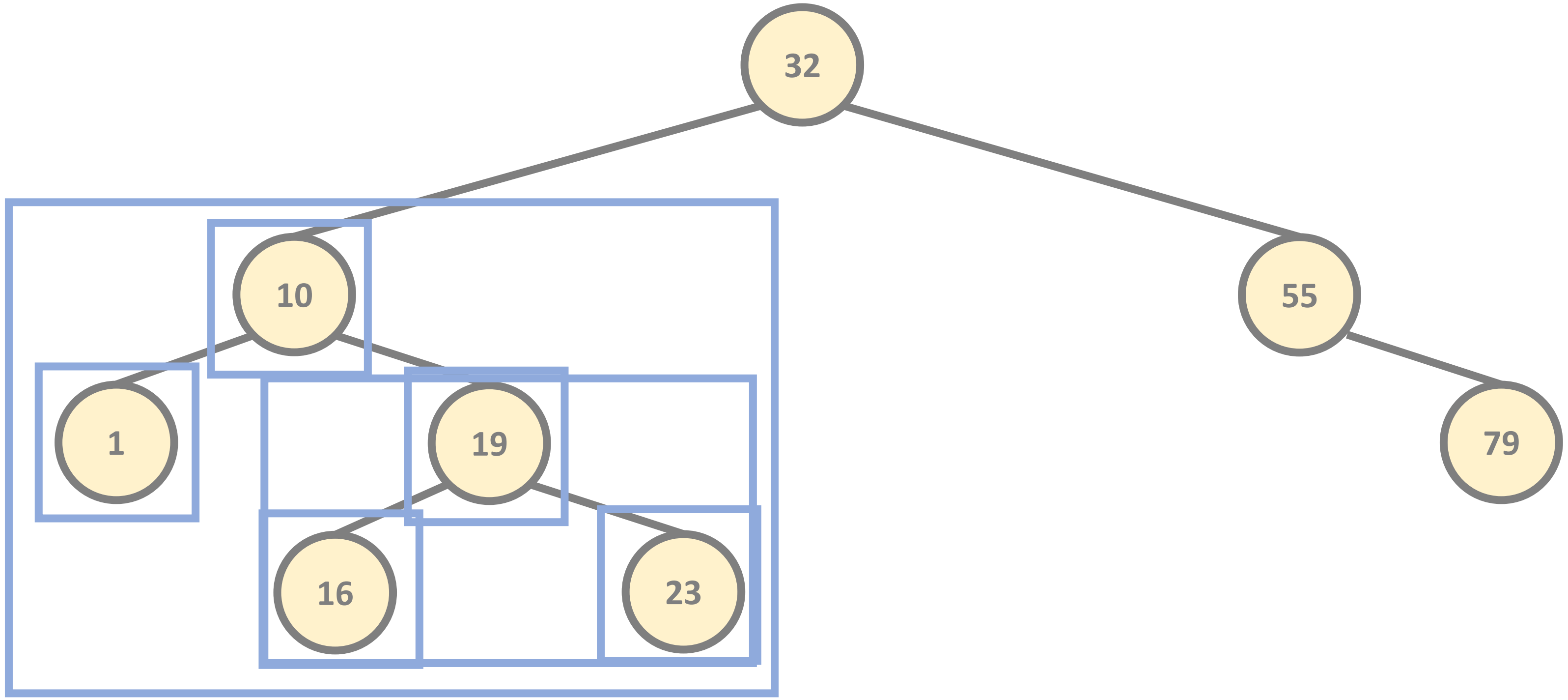
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

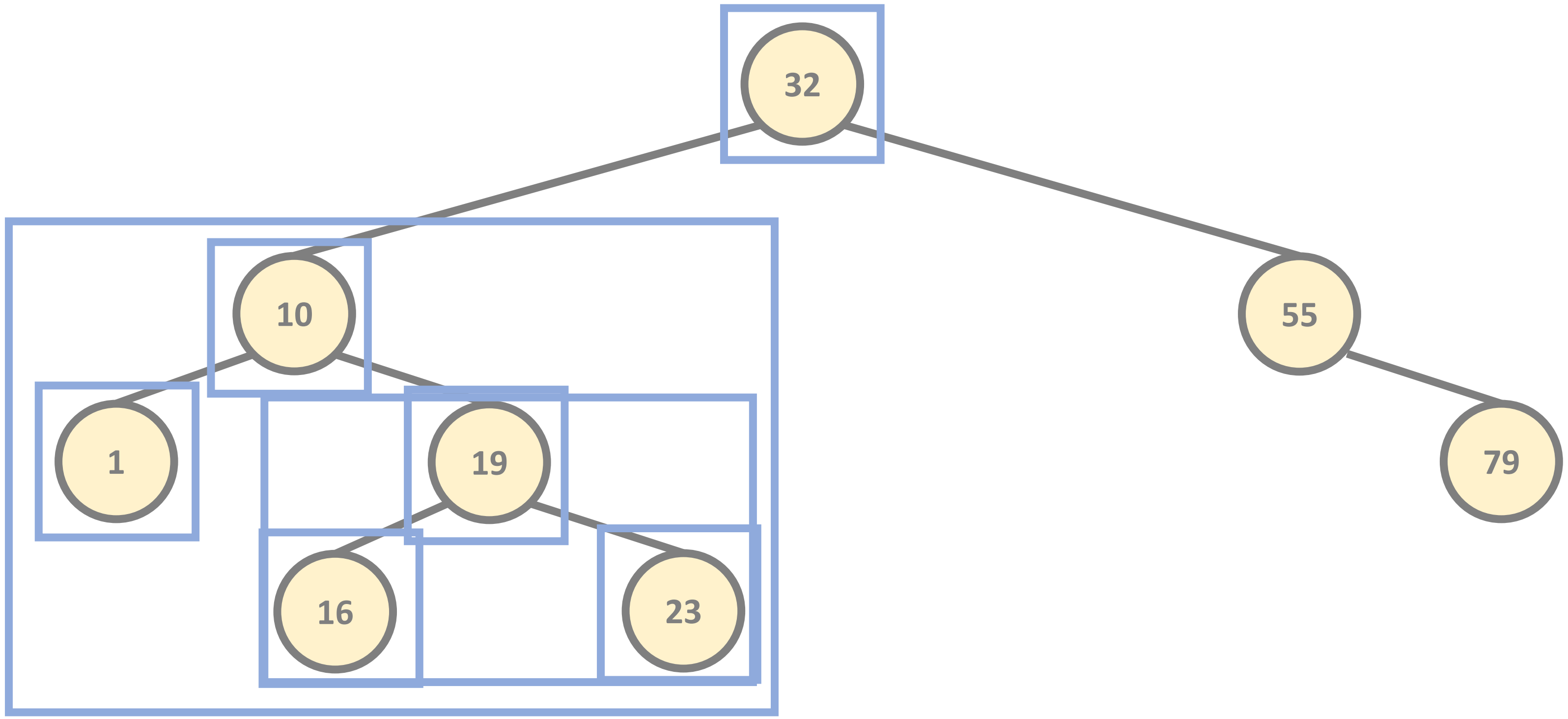
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

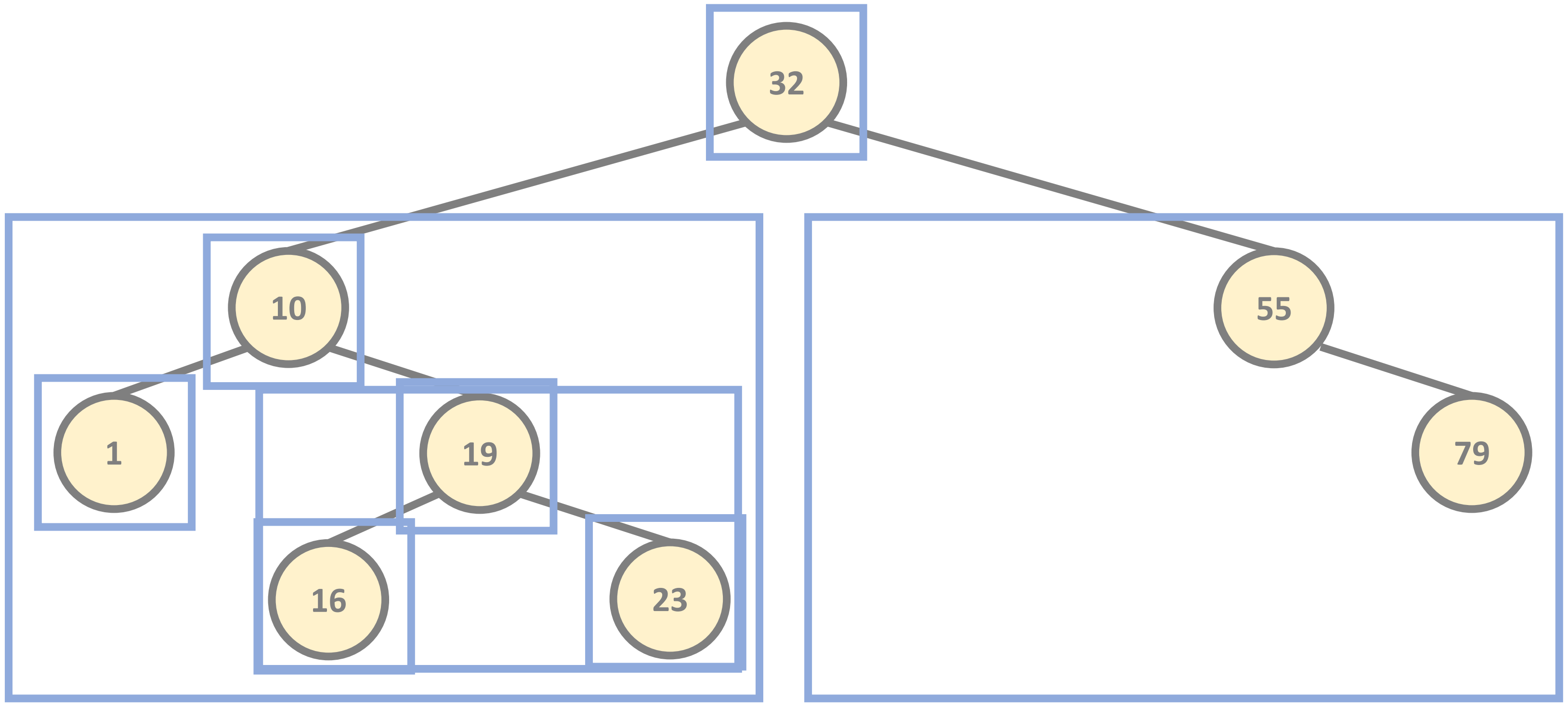
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

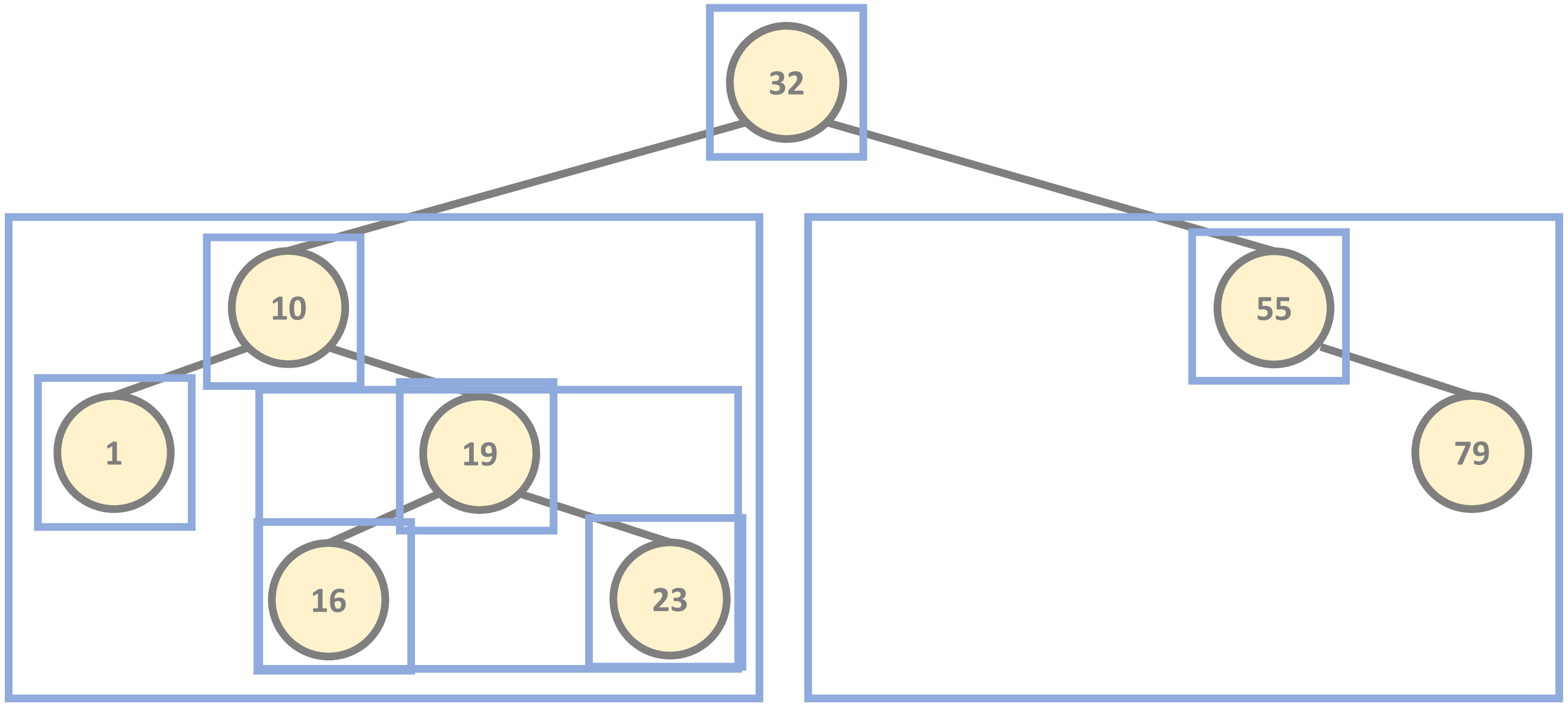
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

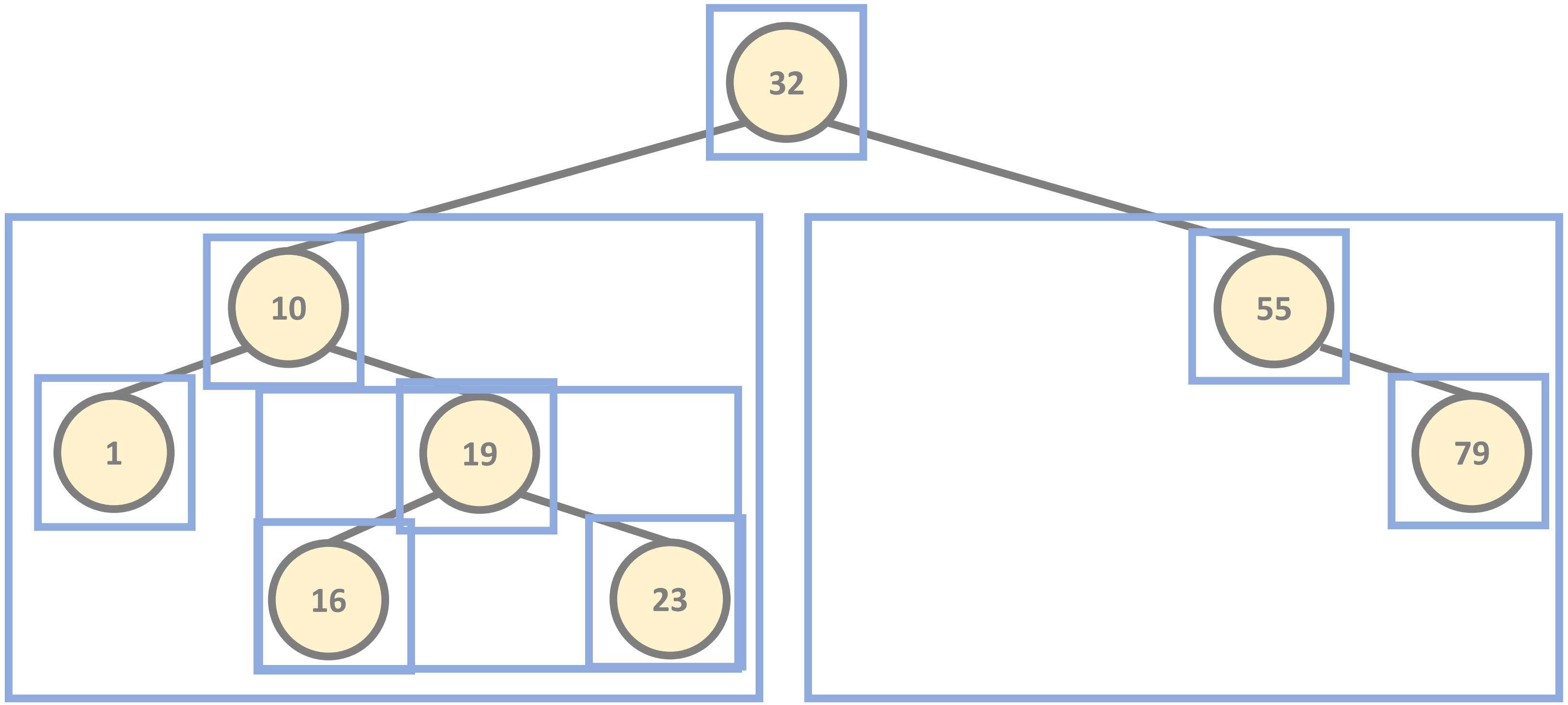
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

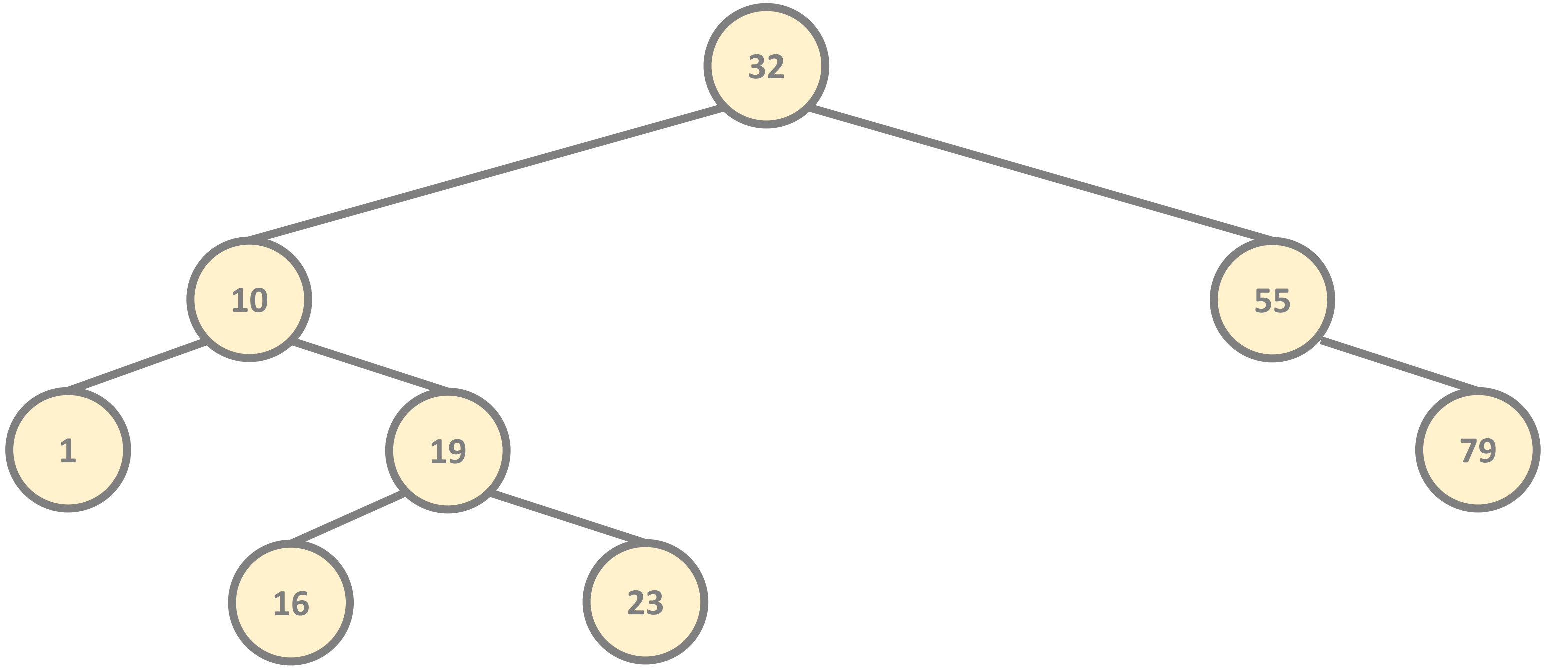
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

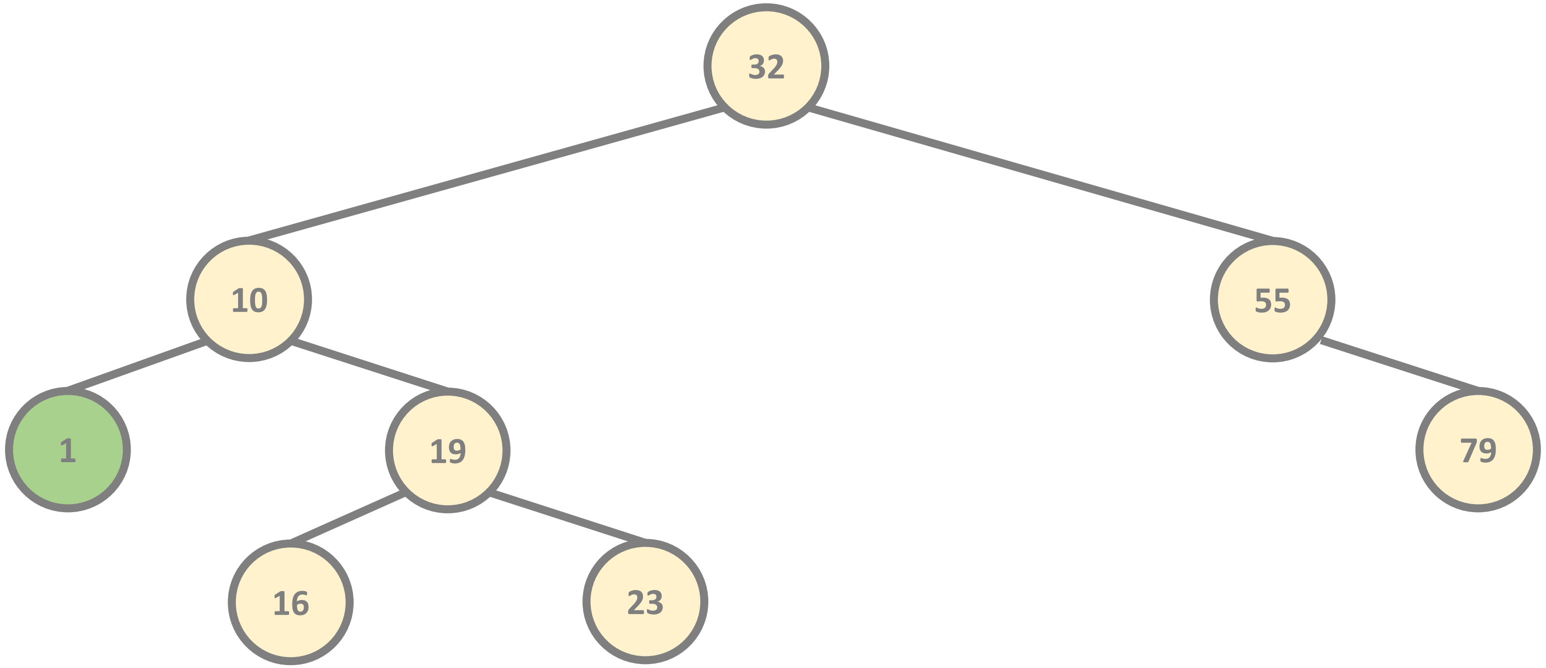
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

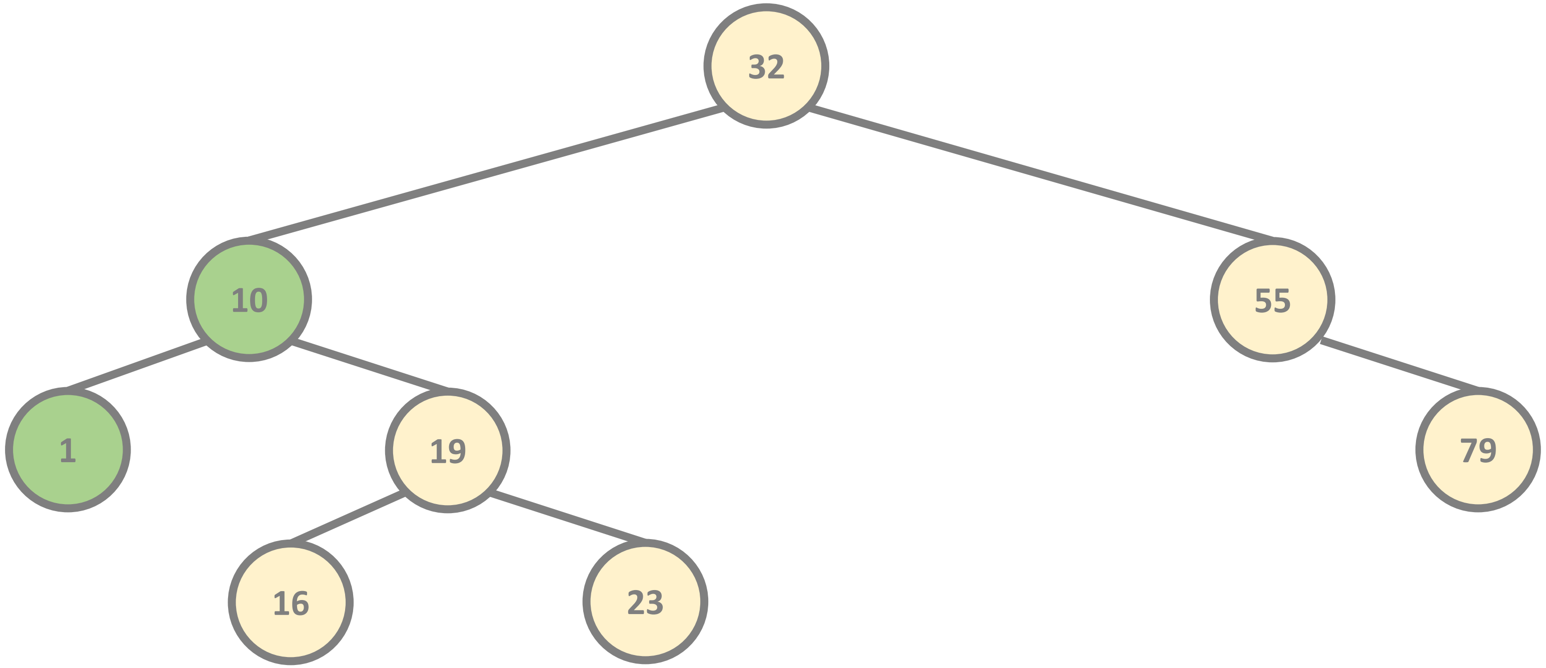
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

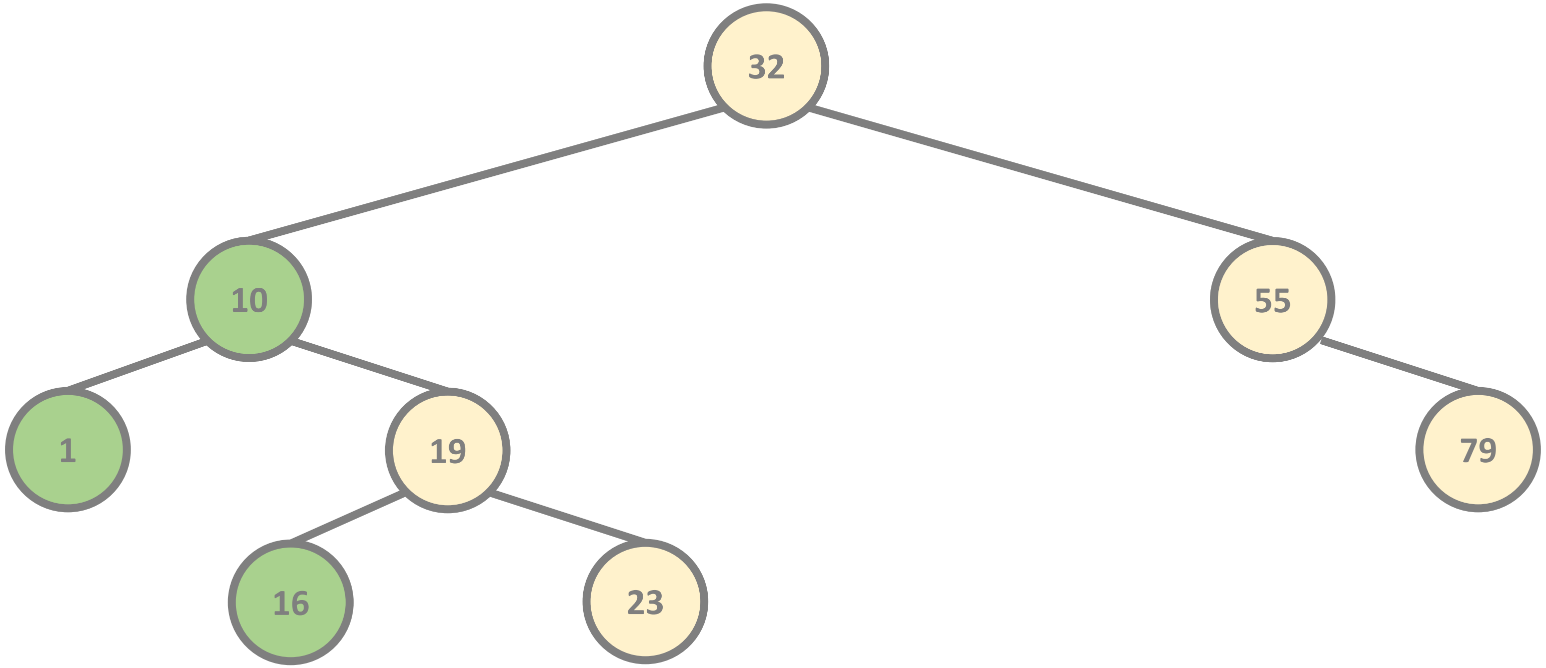
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

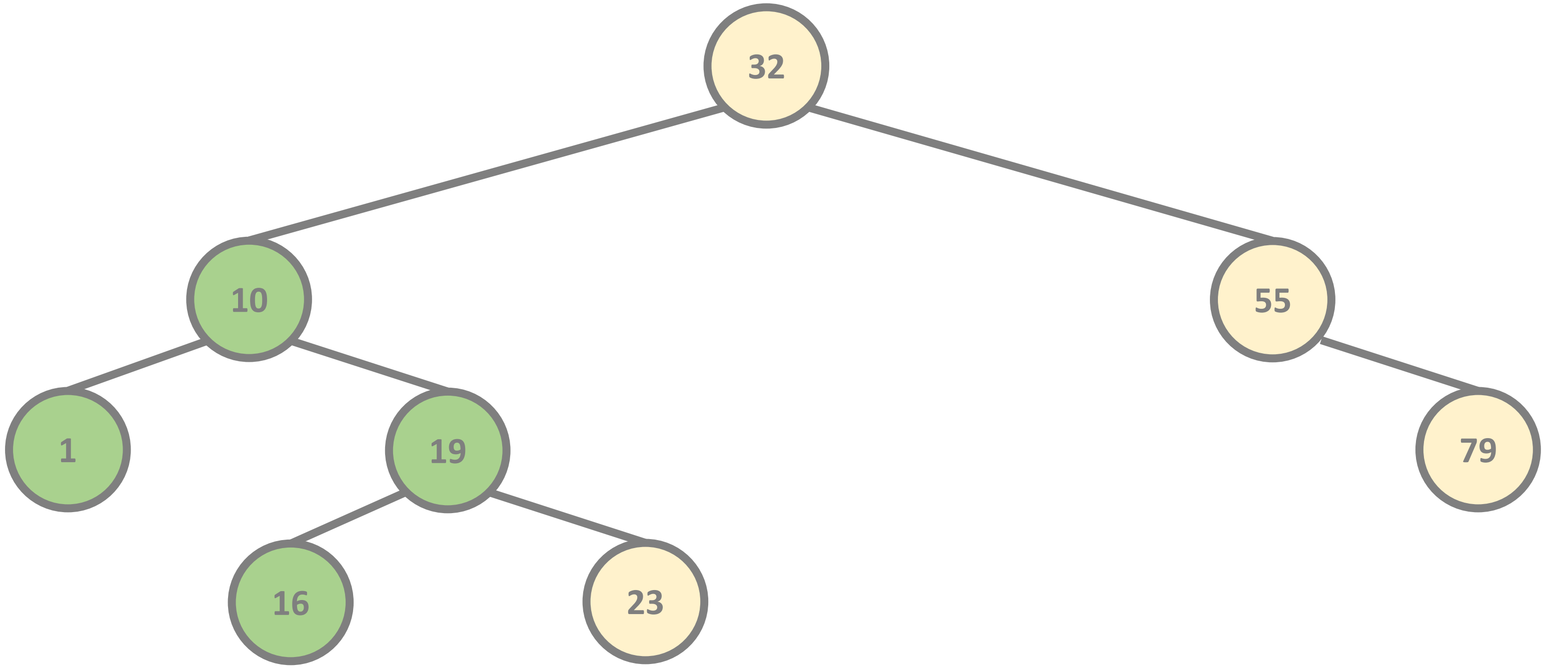
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

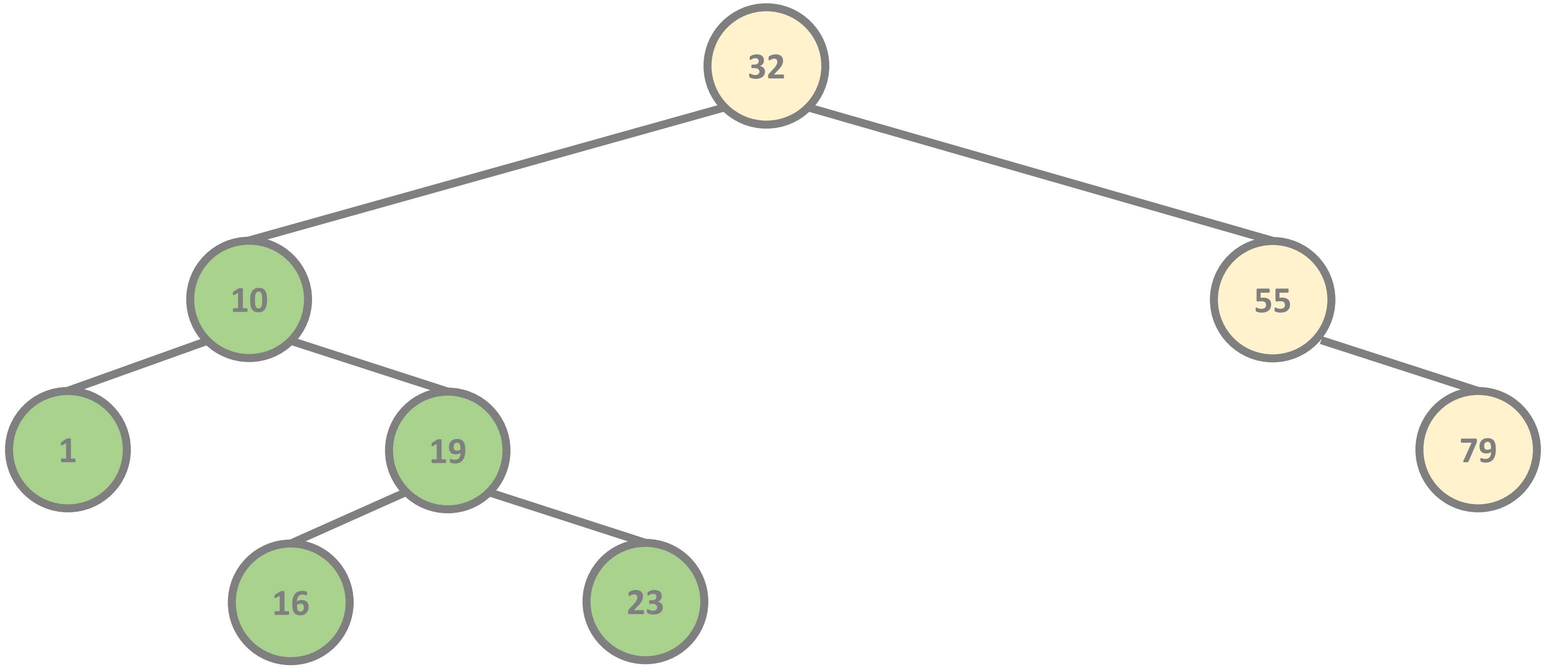
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

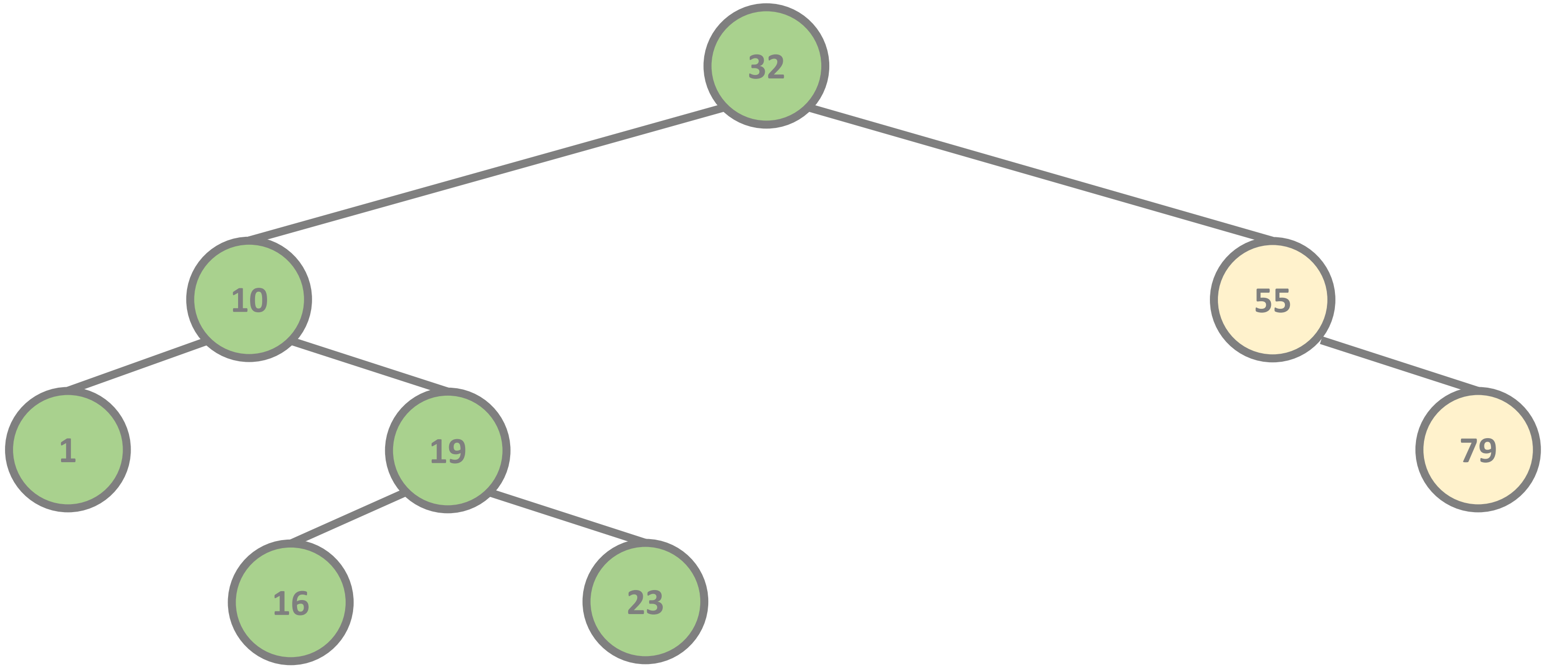
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

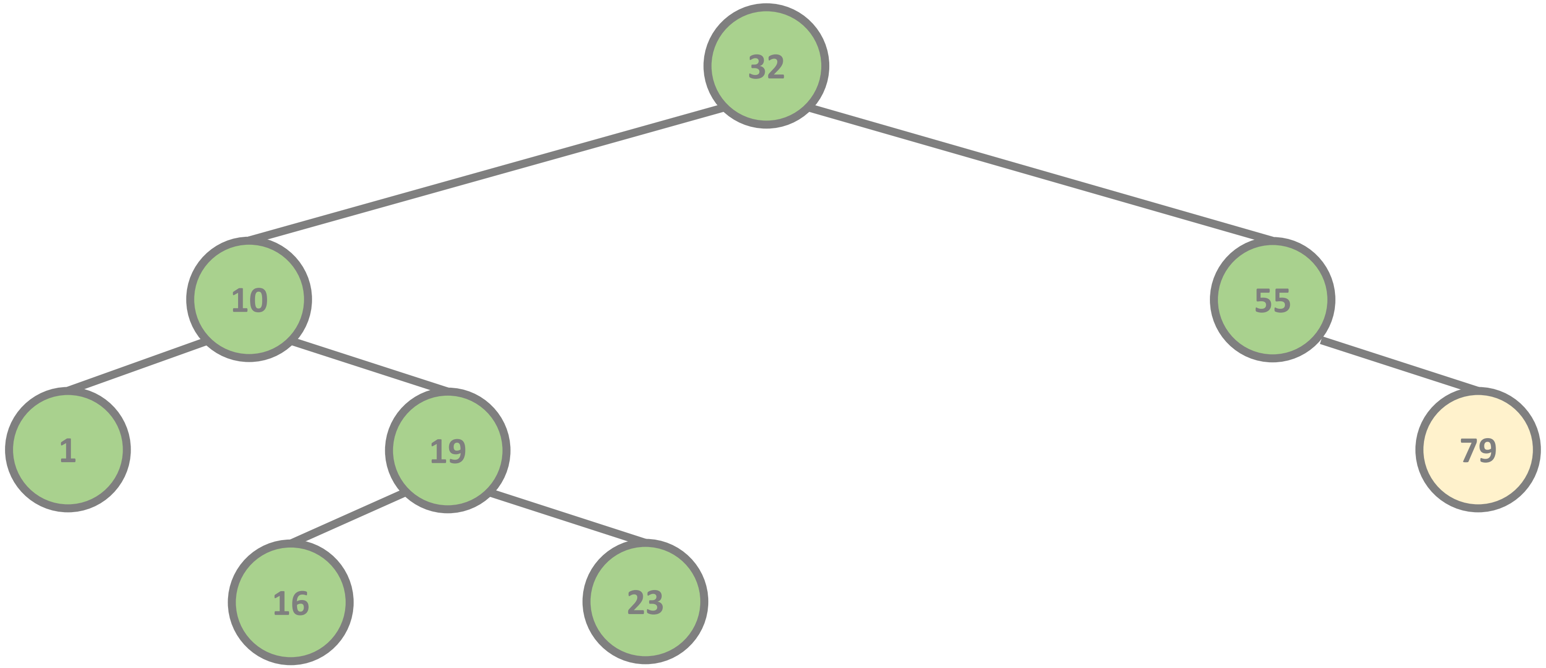
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

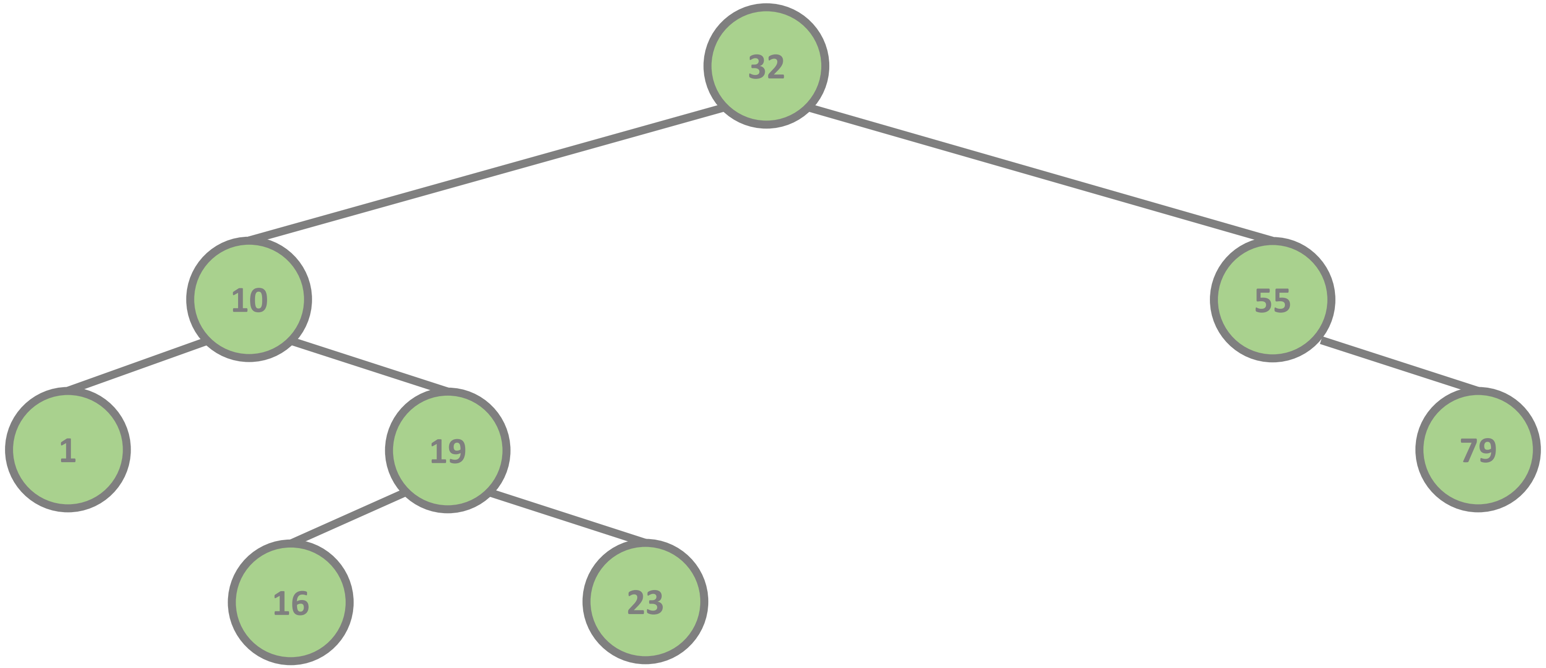
We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Tree Traversal

IN-ORDER TRAVERSAL (SORTED ORDER)

We visit the **left subtree** of the binary tree then the **root node** and finally the **right subtree** in a recursive manner



Binary Search Trees
(Algorithms and Data Structures)

Binary Search Tree

	AVERAGE-CASE	WORST-CASE
space complexity	O(N)	O(N)
insertion	O(logN)	O(N)
deletion (removal)	O(logN)	O(N)
search	O(logN)	O(N)

Binary Search Trees

INSERT(12)

Binary Search Trees

INSERT(12)

Binary Search Trees

Binary Search Trees

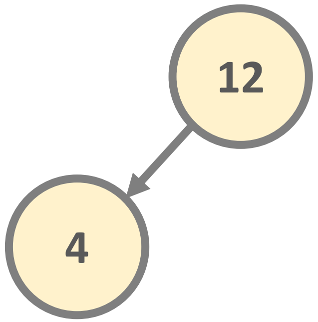
INSERT(4)

Binary Search Trees

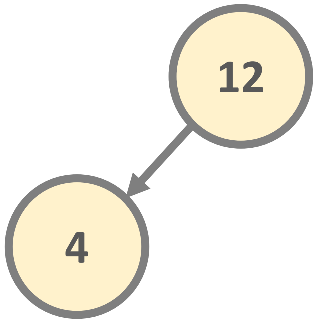
INSERT(4)

Binary Search Trees

INSERT(4)

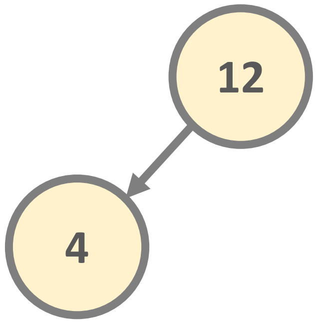


Binary Search Trees



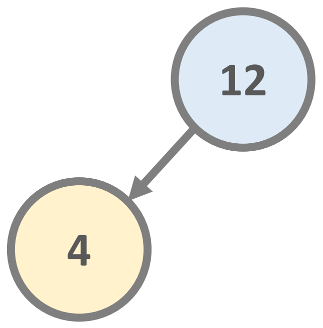
Binary Search Trees

INSERT(3)



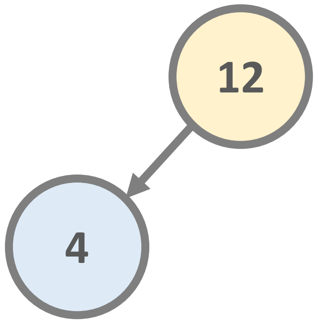
Binary Search Trees

INSERT(3)



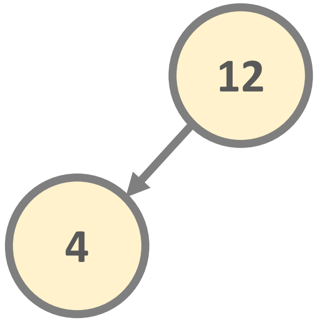
Binary Search Trees

INSERT(3)

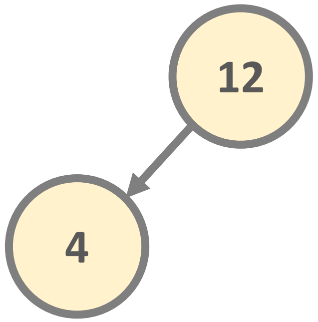


Binary Search Trees

INSERT(3)

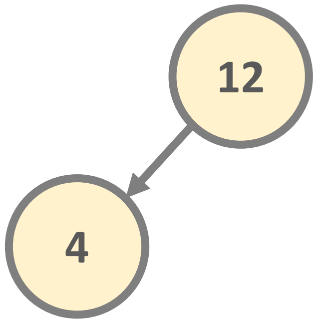


Binary Search Trees



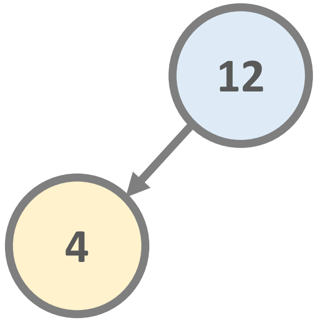
Binary Search Trees

INSERT(3)



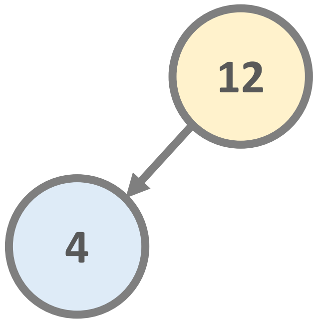
Binary Search Trees

INSERT(3)



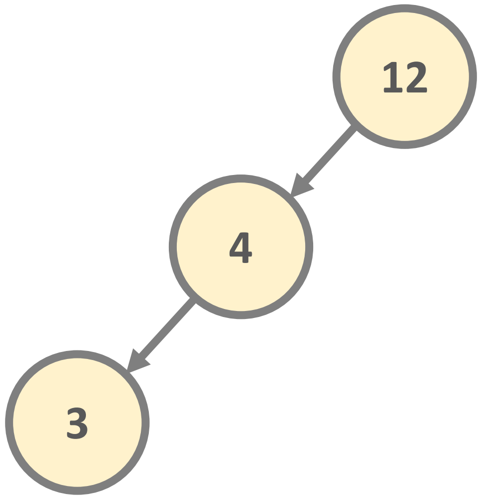
Binary Search Trees

INSERT(3)

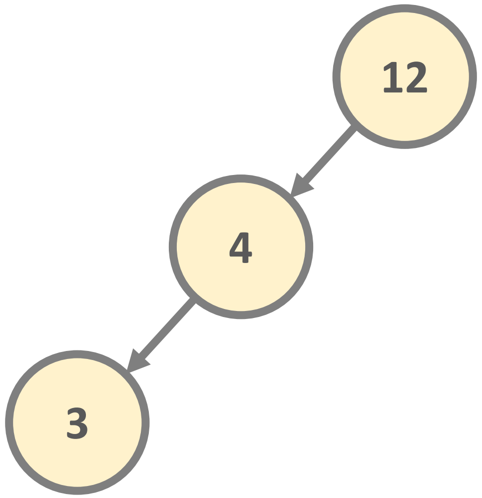


Binary Search Trees

INSERT(3)

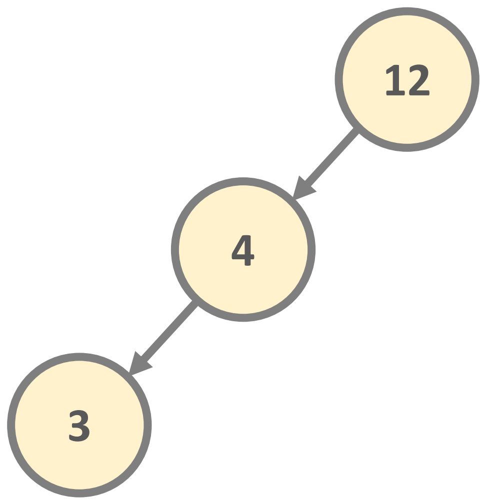


Binary Search Trees



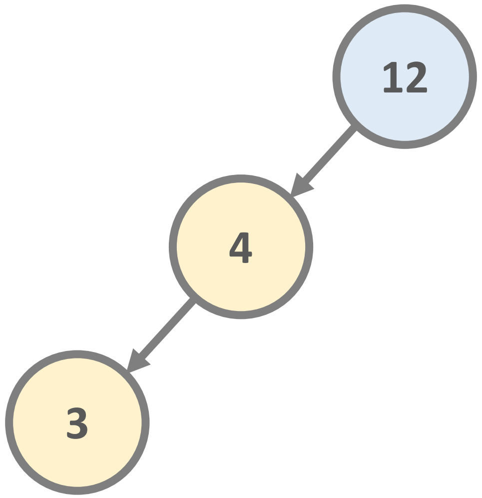
Binary Search Trees

INSERT(2)



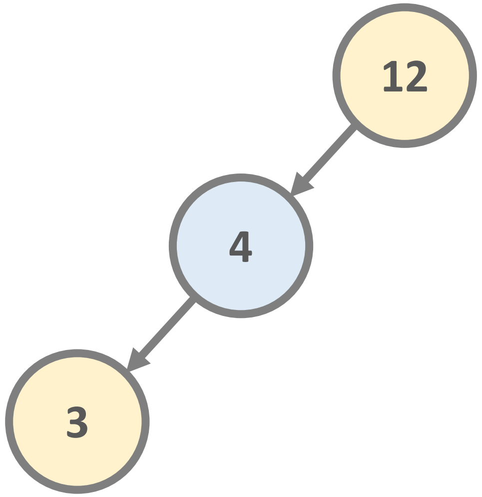
Binary Search Trees

INSERT(2)



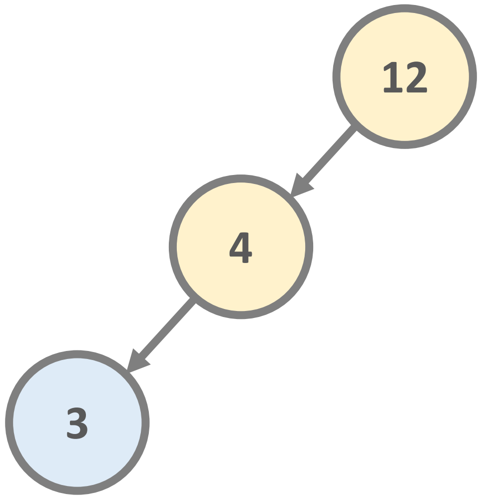
Binary Search Trees

INSERT(2)



Binary Search Trees

INSERT(2)



Binary Search Trees