CSE 3318

Week of 08/07/2023

Instructor : Donna French

Linked List vs Binary Tree

Linked List Node	**Binary Tree Node**

`typedef struct node`
`{`
`    int node_number;`
`    struct node *next_ptr;`
`}`
`NODE;`
 	
`NODE *LinkedListHead;`
	`typedef struct node`
	`{`
	`    int node_number;`
	`    struct node *left_ptr;`
	`    struct node *right_ptr;`
	`}`
	`NODE;`
	
	`NODE *root;`

Binary Tree vs Linked List

Linked List

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->next_ptr = NULL;
```

Binary Tree

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->left_ptr = NULL;
NewNode->right_ptr = NULL;
```

Binary Tree vs Linked List

Add a node to the end of a linked list

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
```

Set the pointer of the last node to the new node

```
TempPtr->next_ptr = NewNode;
```

Add a node to a binary tree

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->left_ptr = NULL;
NewNode->right_ptr = NULL;
```

Set the parent node’s left or right ptr to the address of the new child

```c
/* Allocates memory for a new node with the given data and sets the left and
   right pointers to NULL */
NODE *CreateNewNode(int NodeNumber)
{
      // Allocate memory and assign pointers
      NODE *node = malloc(sizeof(NODE));
      node->left_ptr = NULL;
      node->right_ptr = NULL;

      // Assign data to this node
      node->node_number = NodeNumber;

      printf("Node Number %d %p\n", NodeNumber, node);

      return(node);
}
```

```c
typedef struct node
{
   int node_number;
   struct node *left_ptr;
   struct node *right_ptr;
}
NODE;
```

Reminder!!

When you typedef a structure that contains a pointer to the structure, you must use

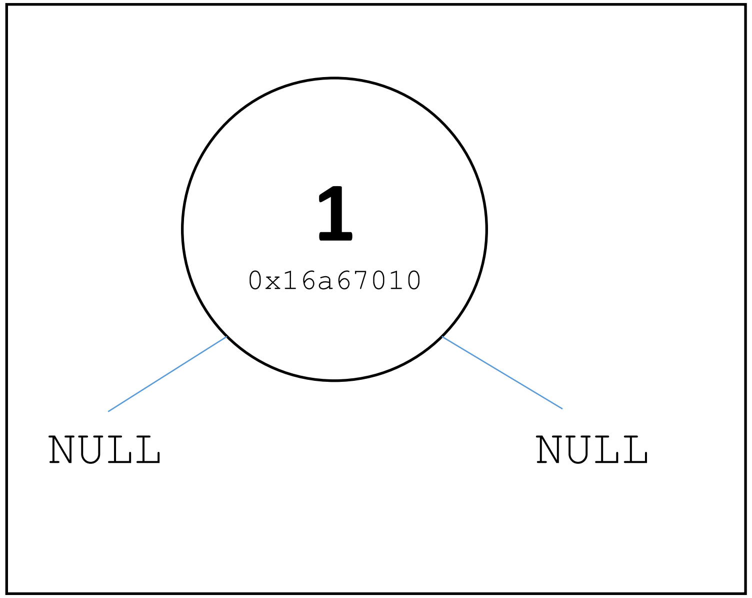
`typedef struct node`

and not just

`typedef struct`

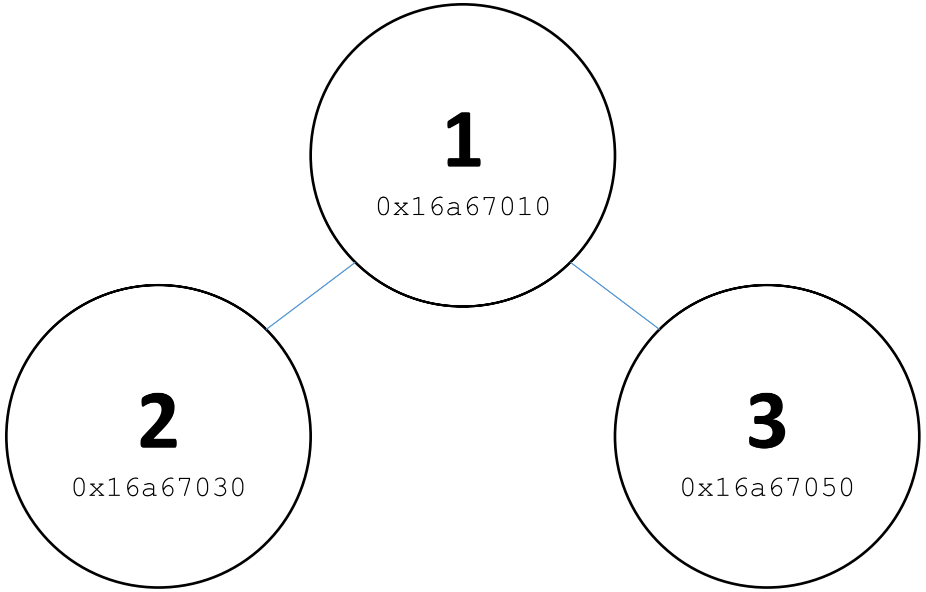
```
Node Number 1 0x16a67010
```

```c
/* declare root of tree */
NODE *root;

/* create root and label with "1" */
root = CreateNewNode(1);


// Print pointer values
printf("\nleft_ptr(1) %p\tright_ptr(1) %p\n",
        root->left_ptr, root->right_ptr);
```




```
left_ptr(1) (nil)        right_ptr(1) (nil)
```

```
root->left_ptr  = CreateNewNode(2);
root->right_ptr = CreateNewNode(3);
```




```
printf("\nleft_ptr(2) %p\tright_ptr(3) %p\n",root->left_ptr, root->right_ptr);
```

```
Node Number 2 0x16a67030
Node Number 3 0x16a67050

left_ptr(2) 0x16a67030  right_ptr(3) 0x16a67050
```

BinaryTreeDemo.c

Binary Tree vs Binary Search Tree

Binary Tree

Each node can have a maximum of two child nodes and there is no order to how the nodes are organized in the tree.

Binary Search Tree

Each node can have a maximum of two child nodes and there is a relative order to how the nodes are organized in the tree.

Binary Search Tree

A binary search tree (with no duplicate node values) has the characteristic that the values in any left subtree are less than the value in its parent node, and the values in any right subtree are greater than the value in its parent node.

The shape of the binary search tree that corresponds to a set of data can vary, depending on the order in which the values are inserted into the tree.

Binary Search Tree

What makes a binary tree a binary search tree?

- All nodes in the left subtree are less than the root

- All nodes in the right subtree are greater than the root

- Each subtree is itself a binary search tree

- No duplicates allowed*

Binary Search Tree (BST)

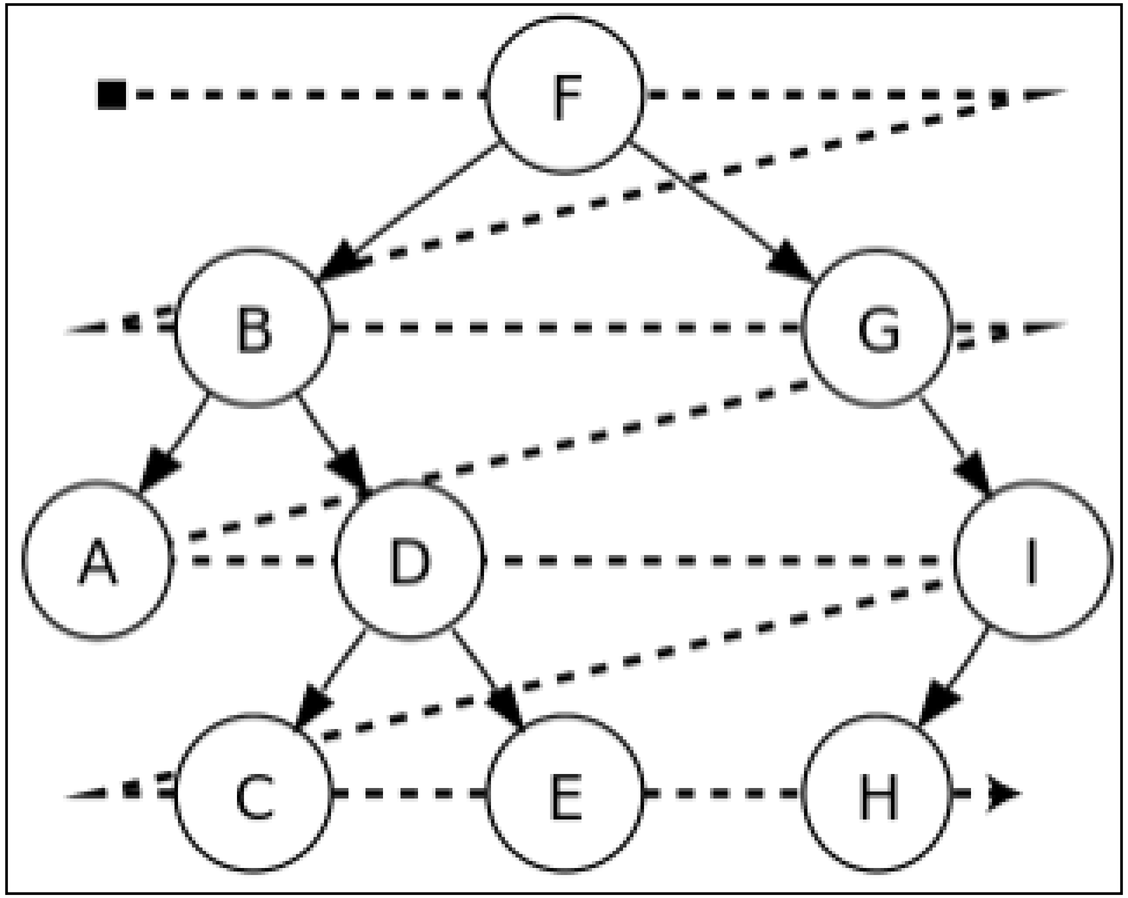
- Linear data structures (linked list, queues and stacks) are traversed in a linear order. Tree structures are traversed in multiple ways - from any given node, there is more than one possible next node in the traversal path

- Tree structures may be traversed in

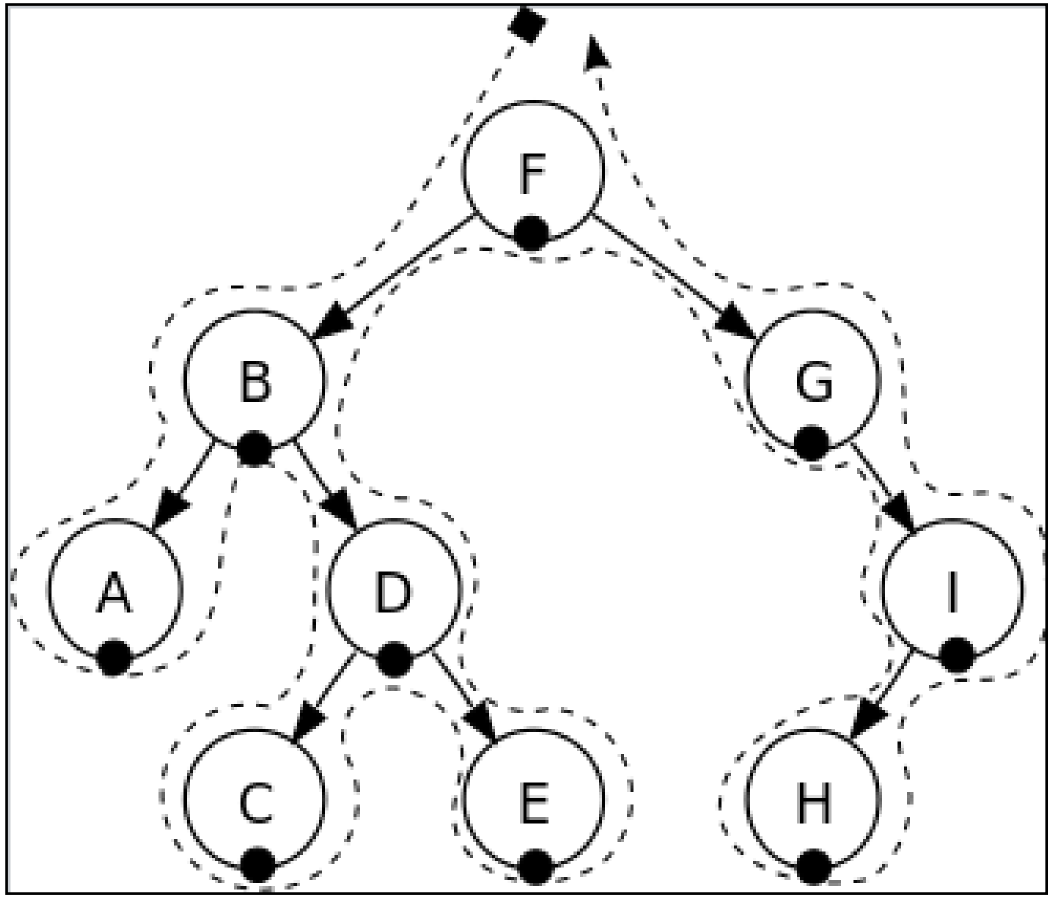
  - Breadth-first Order
  - Depth-first Order

BST Breadth-first vs Depth-first Traversal

Breadth-first



Depth-first



BST Depth-first Traversals

- Inorder Traversal
  - Gives us the nodes in increasing order
- Preorder Traversal
  - Parent nodes are visited before any of its child nodes
  - Used to create a copy of the tree
  - File systems use it to track your movement through directories
- Postorder Traversal
  - Used to delete the tree
  - File systems use it to delete folders and the files under them

BST Depth-first Traversals

Depth First Tree Traversals

Preorder

Root, Left, Right

4 2 1 3 5

Postorder

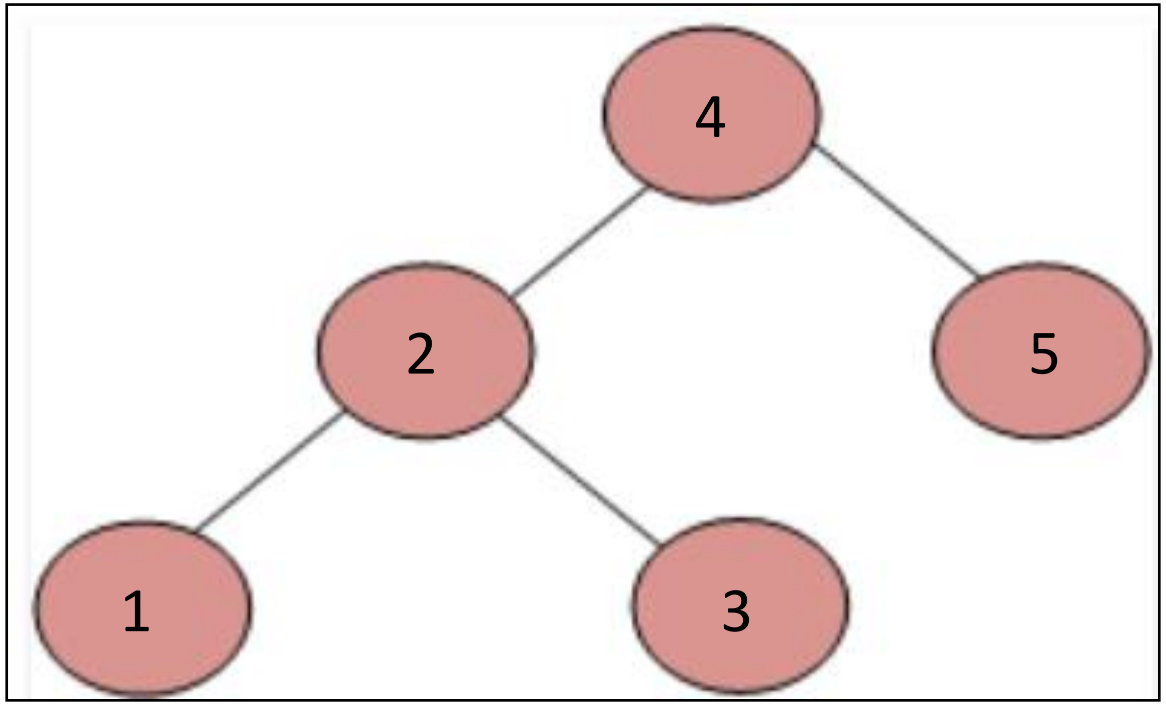
Left, Right, Root

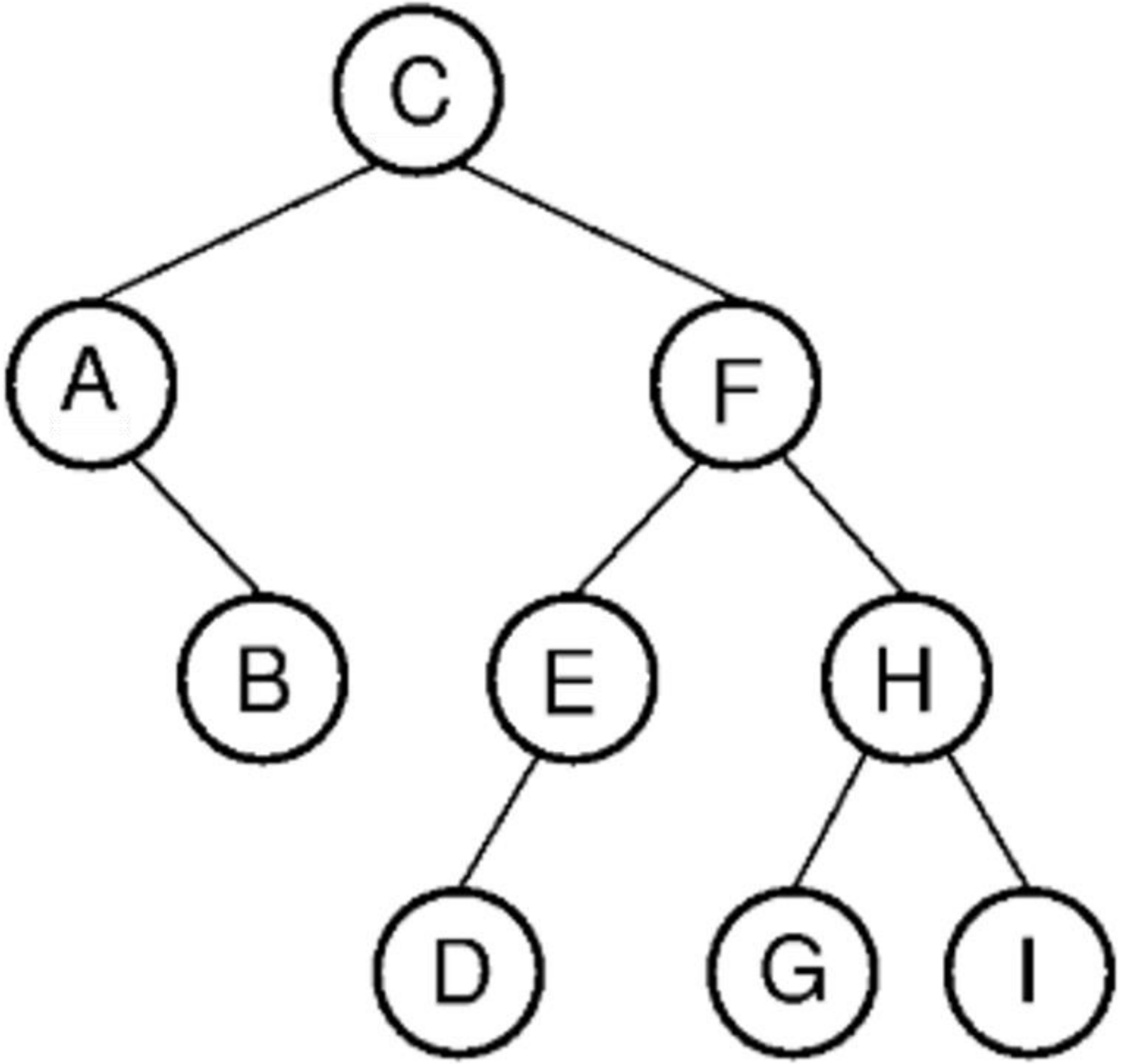
1 3 2 5 4

Inorder

Left, Root, Right

1 2 3 4 5





Depth First Tree Traversals

Preorder

Root, Left, Right

C A B F E D H G I

Postorder

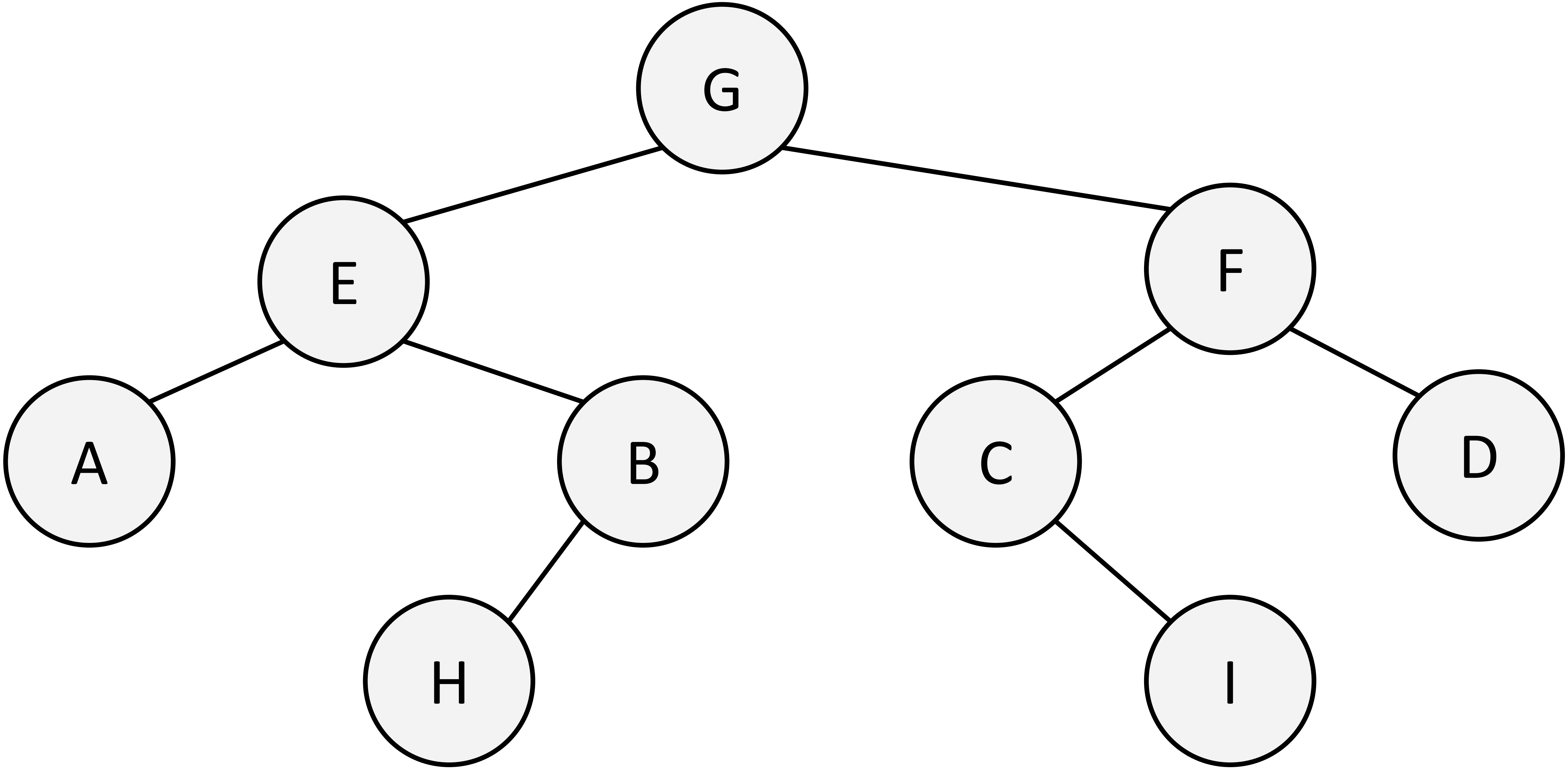
Left, Right, Root

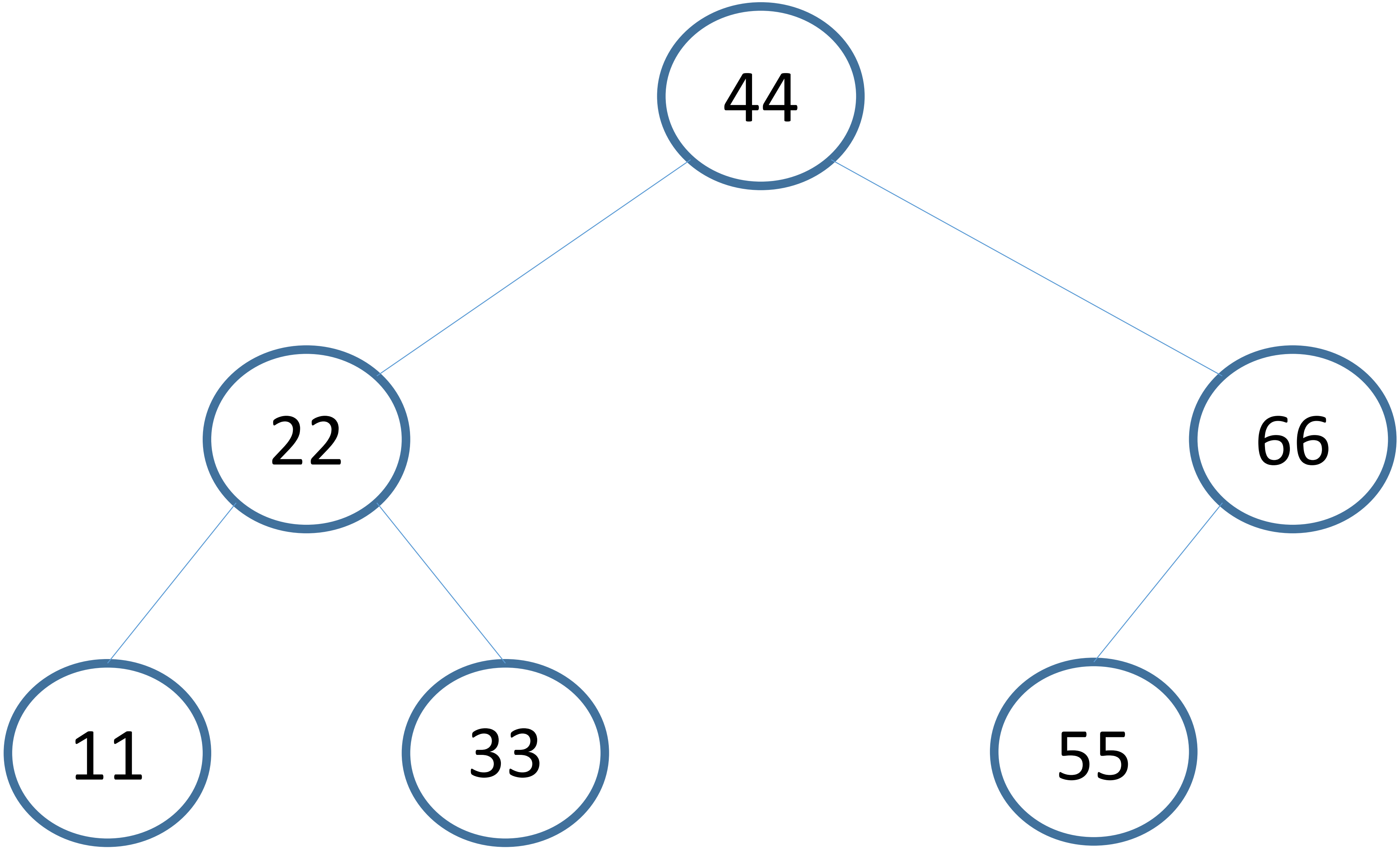
B A D E G I H F C

Inorder

Left, Root, Right

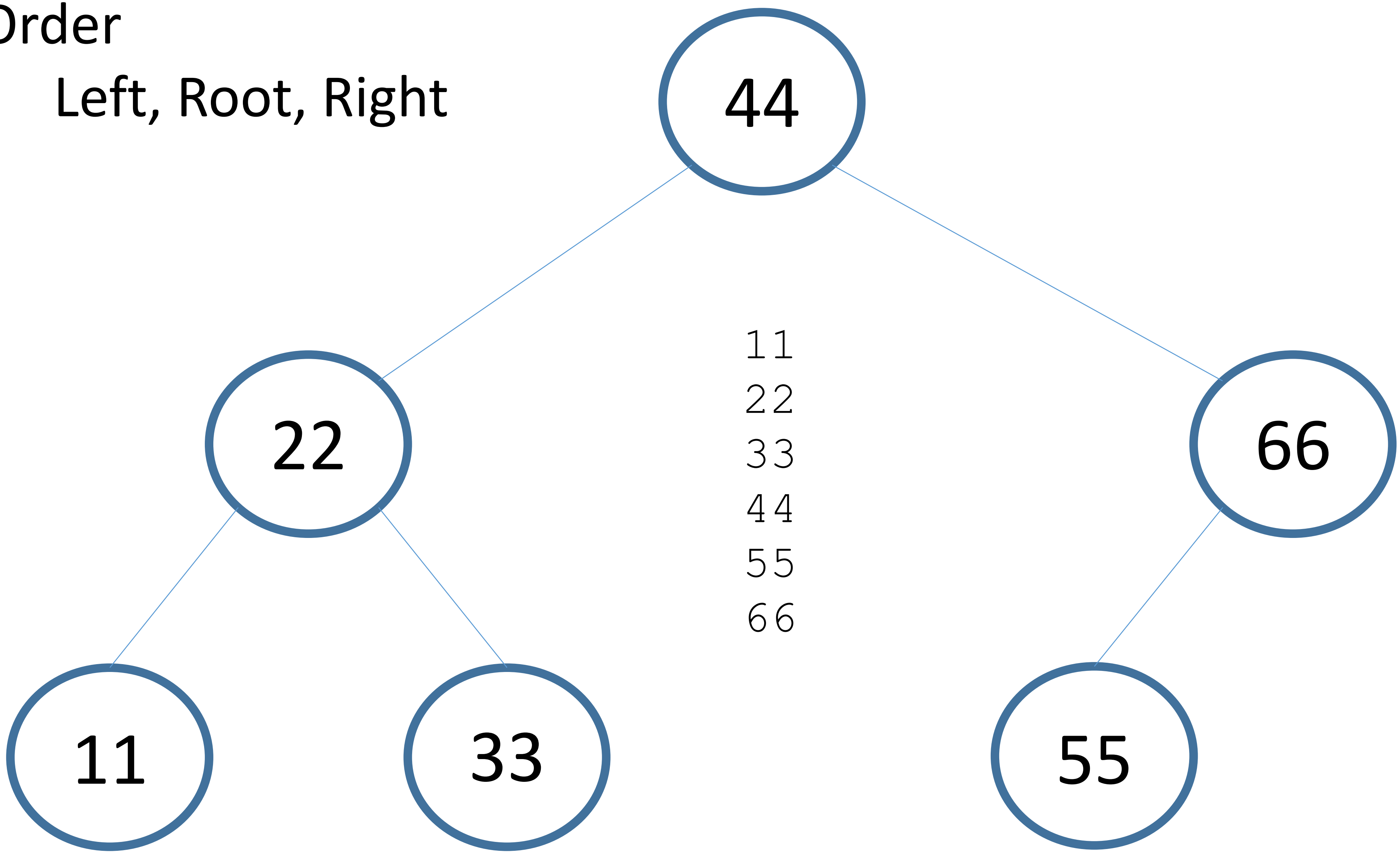
A B C D E F G H I


G
E
F
A
B
C
D
H
I


44
22
66
11
33
55

In Order

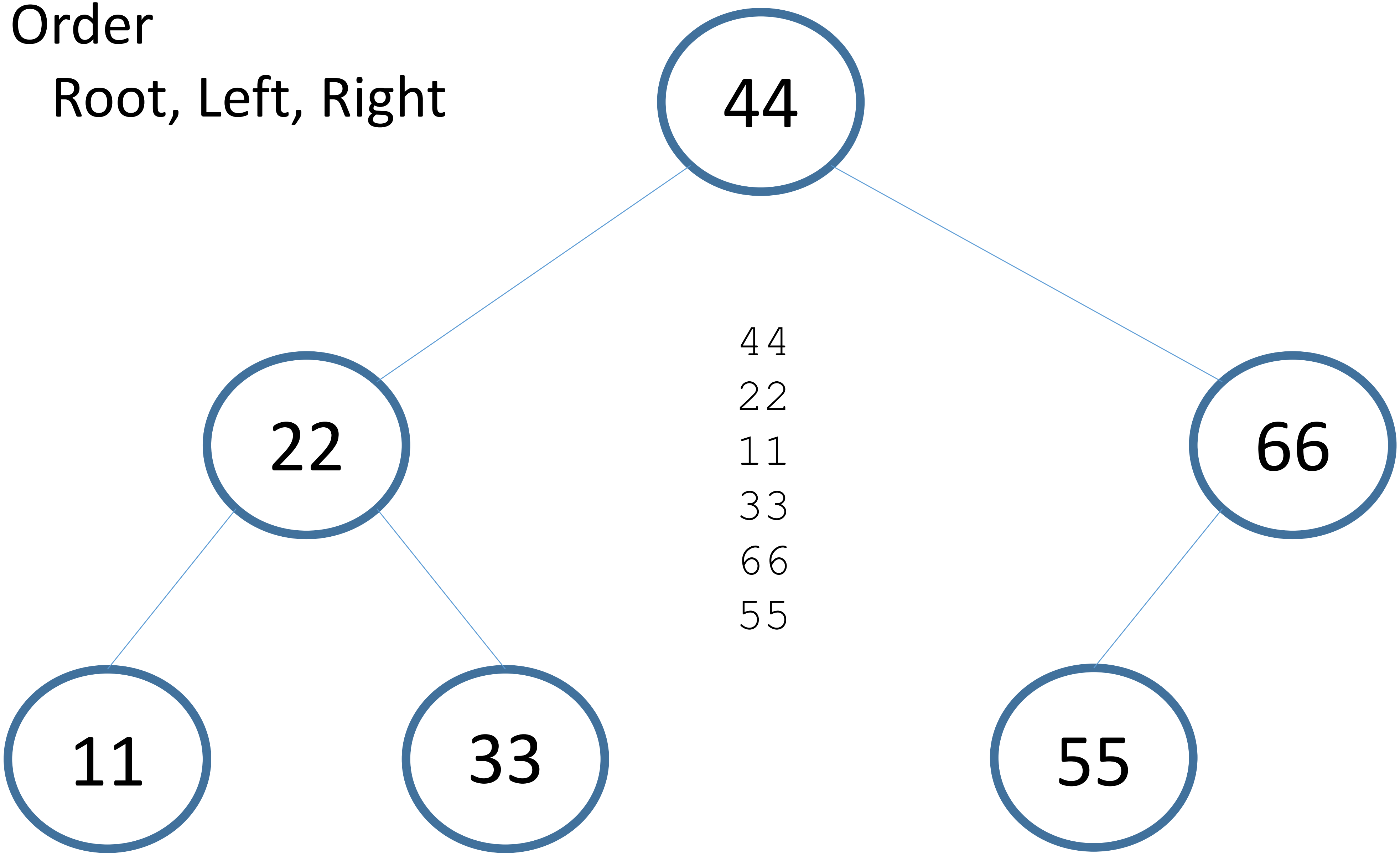
Left, Root, Right



11
22
33
44
55
66

Pre Order

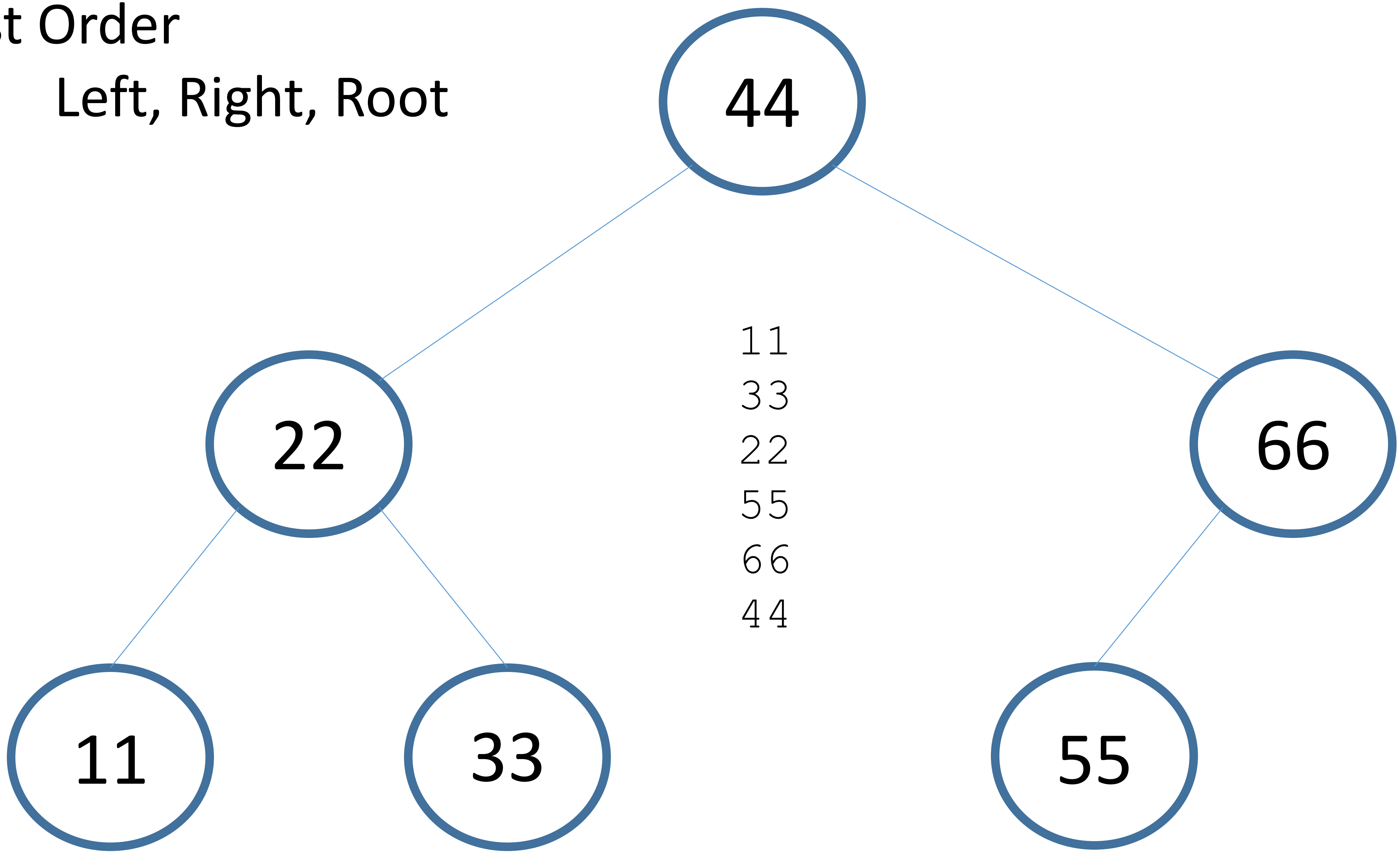
Root, Left, Right



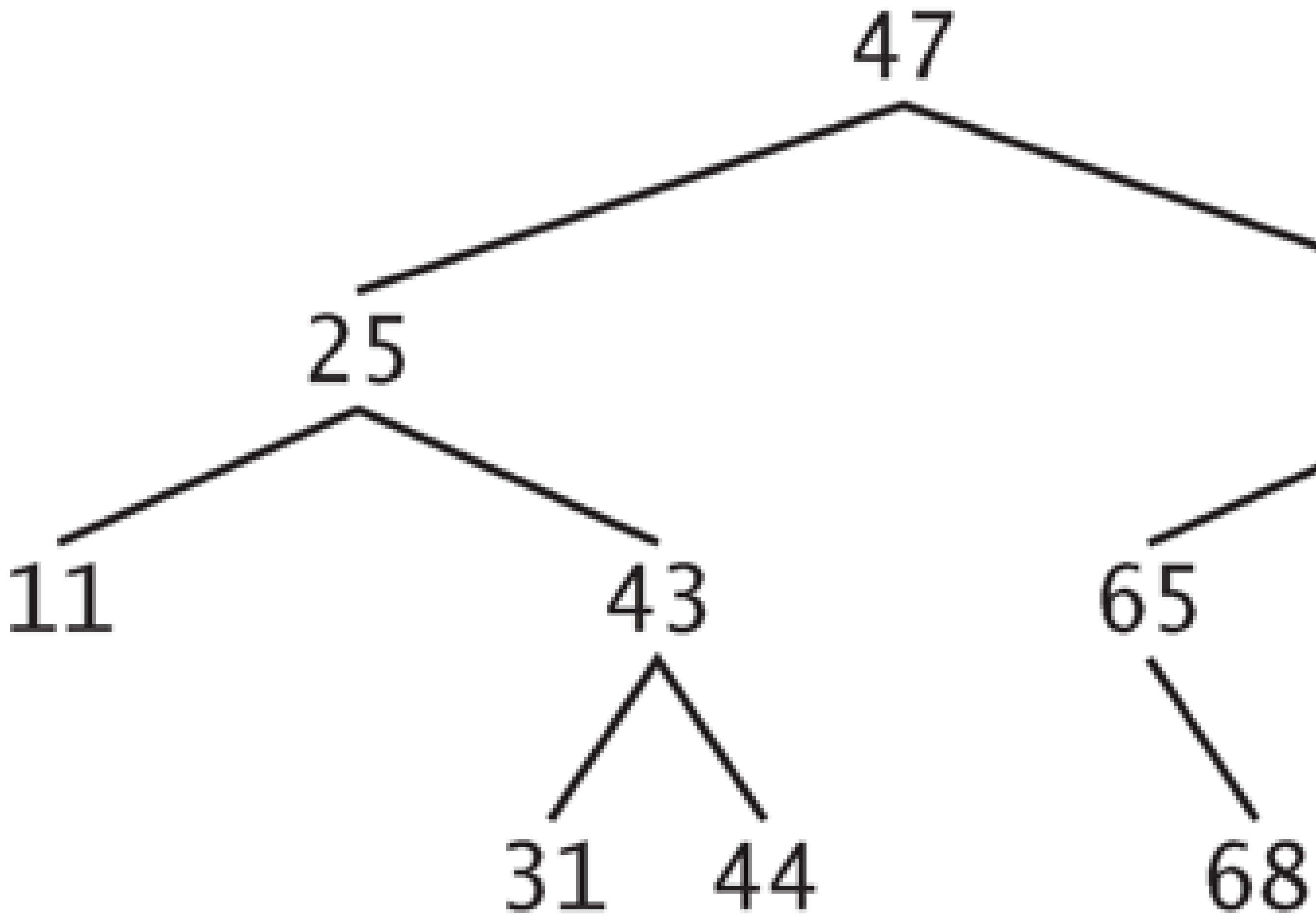
44
22
11
33
66
55

Post Order

Left, Right, Root



11
33
22
55
66
44


47
25
77
11
43
65
31
44
68

```
How many nodes in the tree? 9

Enter data for node 1 : 47

Enter data for node 2 : 25

Enter data for node 3 : 77

Enter data for node 4 : 11

Enter data for node 5 : 43

Enter data for node 6 : 65

Enter data for node 7 : 31

Enter data for node 8 : 44

Enter data for node 9 : 68
```

BST Traversal in Inorder

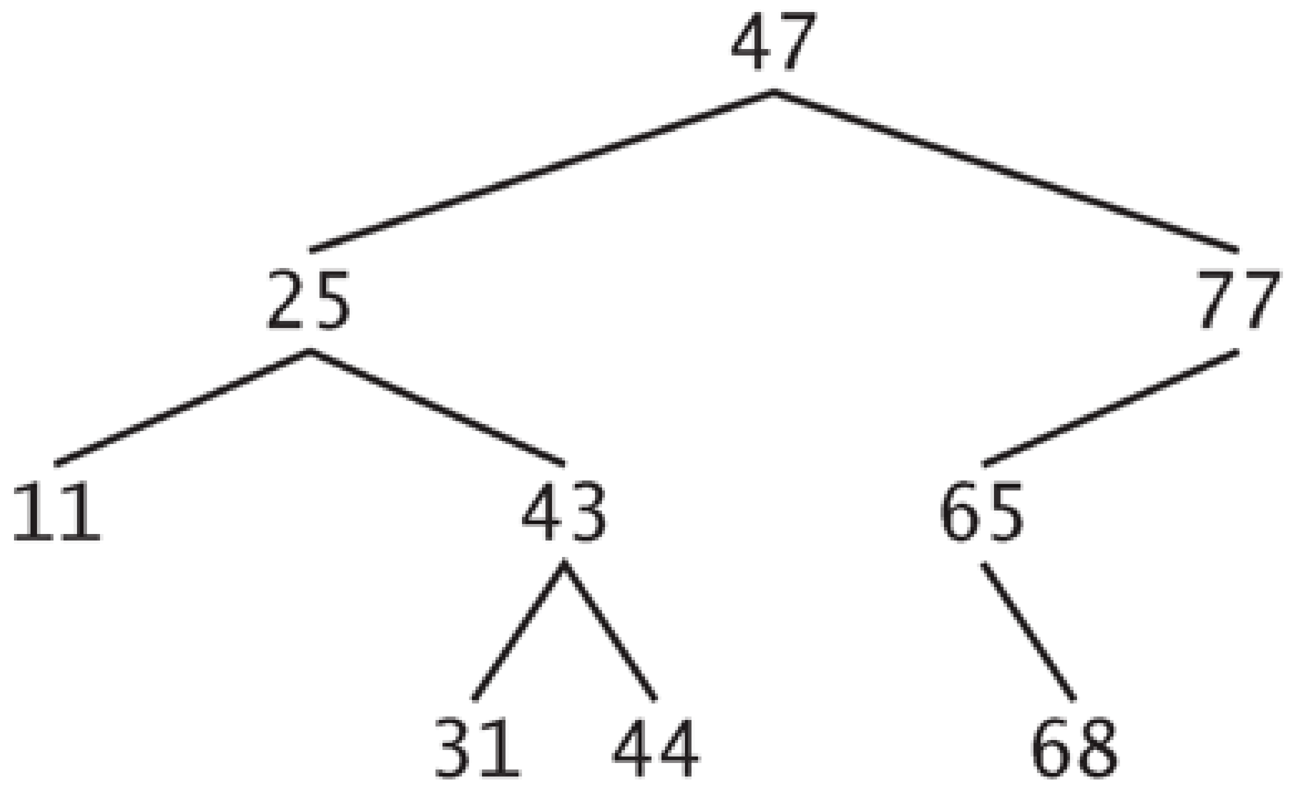
```
Node4-11  Node2-25  Node7-31
Node5-43  Node8-44  Node1-47
Node6-65  Node9-68  Node3-77
```

BST Traversal in Preorder

```
Node1-47  Node2-25  Node4-11
Node5-43  Node7-31  Node8-44
Node3-77  Node6-65  Node9-68
```

BST Traversal in Postorder

```
Node4-11  Node7-31  Node8-44
Node5-43  Node2-25  Node9-68
Node6-65  Node3-77  Node1-47
```



```
typedef struct node
{
    int node_data;
    struct node *right;
    struct node *left;
}
NODE;


NODE *root = NULL;


AddBSTNode(&root, node_data);
```

```c
void AddBSTNode(NODE **current_node, int add_data)
{
    if (*current_node == NULL)
    {
        *current_node = malloc(sizeof(NODE));
        (*current_node)->left = (*current_node)->right = NULL;
        (*current_node)->node_data = add_data;
    }
    else
    {
        if (add_data < (*current_node)->node_data )
            AddBSTNode(&(*current_node)->left, add_data);

        else if(add_data > (*current_node)->node_data )
            AddBSTNode(&(*current_node)->right, add_data);

        else
            printf(" Duplicate Element !! Not Allowed !!!");
    }
}
```

```c
Inorder(root);

Preorder(root);

Postorder(root);
```

```c
void Inorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      Inorder(tree_node->left);
      printf("Node%d",
              tree_node->node_data);
      Inorder(tree_node->right);
   }
}
```

```c
void Preorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      printf("Node%d",
              tree_node->node_data);
      Preorder(tree_node->left);
      Preorder(tree_node->right);
   }
}
```

```c
void Postorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      Postorder(tree_node->left);
      Postorder(tree_node->right);
      printf("Node%d",
              tree_node->node_data);
   }
}
```

Priority Queue

Making a priority queue

Use a heap structure

Insert new elements into the heap

Remove the top element to get the highest priority element

Change priorities by removing the element and then re-inserting it

Fibonacci Heap

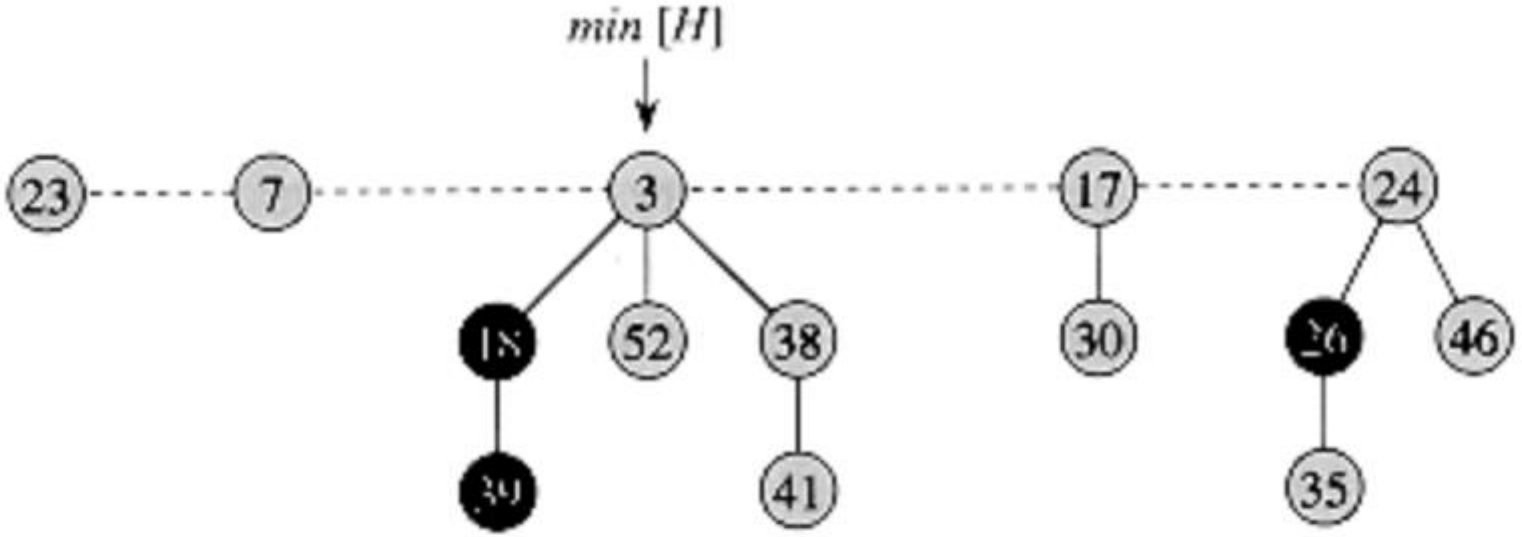
A fibonacci heap is a data structure that consists of a collection of trees which follow min heap or max heap property.

In a fibonacci heap, a node can have more than two children or no children at all.

Fibonacci heaps are linked lists of heap ordered trees.

A pointer to the minimum element is maintained

Fibonacci Heap



Fibonacci Heap

The root nodes of a Fibonacci heap are connected with a circular doubly linked list.

Each layer of the Fibonacci heap are connected with circular doubly linked list.

Parent and child nodes are connected with circular doubly linked list.

Fibonacci Heap

A node can be removed from a circular doubly linked list in a Fibonacci heap in O(1) time.

Two of these types of heaps can be concatenated in O(1) time

An item is added to a Fibonacci Heap by creating a new heap and connecting to the circular doubly linked list of roots.

The pointer to the minimum element is updated if needed when a new item is added.

Nodes are "marked" when they lose a child node and are moved up to the root level when they lose another child node.

Algorithm Interview Questions & Answers

Question : What is an algorithm? What is the need for an algorithm?

An algorithm is a well-defined computational procedure that takes some values or the set of values, as an input and produces a set of values or some values, as an output.

Algorithms help us measure and analyze the complexity time and space of the problems.

We can compare the performance of the algorithms with respect to other techniques.

Algorithm Interview Questions & Answers

Algorithms provide the basic idea of the problem and an approach to solve it.

Some reasons to use algorithms are...

- improves the efficiency of an existing technique.
- gives a strong description of requirements and goal of the problems to the designer.
- provides a reasonable understanding of the flow of the program.
- measures the performance of the methods in different cases (Best cases, worst cases, average cases).
- identifies the resources (input/output, memory) cycles required by the algorithm.
- reduces the cost of design

Algorithm Interview Questions & Answers

Question : What is the Complexity of Algorithm?

The complexity of the algorithm is a way to classify how efficient an algorithm is compared to alternative ones. Its focus is on how execution time increases with the data set to be processed. The computational complexity of the algorithm is important in computing.

It is very suitable to classify algorithm based on the relative amount of time or relative amount of space they required and specify the growth of time/ space requirement as a function of input size.

Algorithm Interview Questions & Answers

Time complexity

Time complexity is the running time of a program as a function of the size of the input.

Space complexity

Space complexity analyzes the algorithm, based on how much space an algorithm needs to complete its task. Space complexity analysis was critical in the early days of computing (when storage space on the computer was limited) – not as important today.

Algorithm Interview Questions & Answers

Worst-case: f(n)

It is defined by the maximum number of steps taken on any instance of size n.

Best-case: f(n)

It is defined by the minimum number of steps taken on any instance of size n.

Average-case: f(n)

It is defined by the average number of steps taken on any instance of size n.

Algorithm Interview Questions & Answers

Write an algorithm to reverse a string. For example, if my string is "French" then my result will be "hcnerF".

Step1: Create two variables `i` and `j`

Step2: set `i` equal to 0 and set `j` equal to the length of the string - 1

Step3: swap string [i] with string[j]

Step4: increment `i` by 1 and decrement `j` by 1

Step5: repeat steps 3 and 4 while $i < j$

Algorithm Interview Questions & Answers

Write an algorithm to insert a node in a sorted linked list.

What are the 2 possibilities?

Linked list is empty

Linked list is not empty

Algorithm Interview Questions & Answers

Linked list is empty

Create a new node

If linked list is empty (linked list head is NULL), then set linked list head to the new node.

Algorithm Interview Questions & Answers

Linked list is not empty

Create a new node

Traverse the linked list until you find where to insert the new node.

Save the previous node's next pointer. Set previous node's pointer to the address of the new node. Set the new node's next pointer to the saved address that was saved from previous node's next pointer.

Algorithm Interview Questions & Answers

Add new node to linked list

Create a new node

Traverse the linked list to the end.

Set the next pointer of the end node to the new node's address.

Algorithm Interview Questions & Answers

What are the Asymptotic Notations?

Asymptotic analysis is used to measure the efficiency of an algorithm that doesn't depend on machine-specific constants and prevents the algorithm from comparing the time taking algorithm.

Asymptotic notation is a mathematical tool that is used to represent the time complexity of algorithms for asymptotic analysis.

Algorithm Interview Questions & Answers

What are the Asymptotic Notations?

Asymptotic analysis is used to measure the efficiency of an algorithm that doesn't depend on machine-specific constants and prevents the algorithm from comparing the time taking algorithm.

Asymptotic notation is a mathematical tool that is used to represent the time complexity of algorithms for asymptotic analysis.

Algorithm Interview Questions & Answers

θ Notation

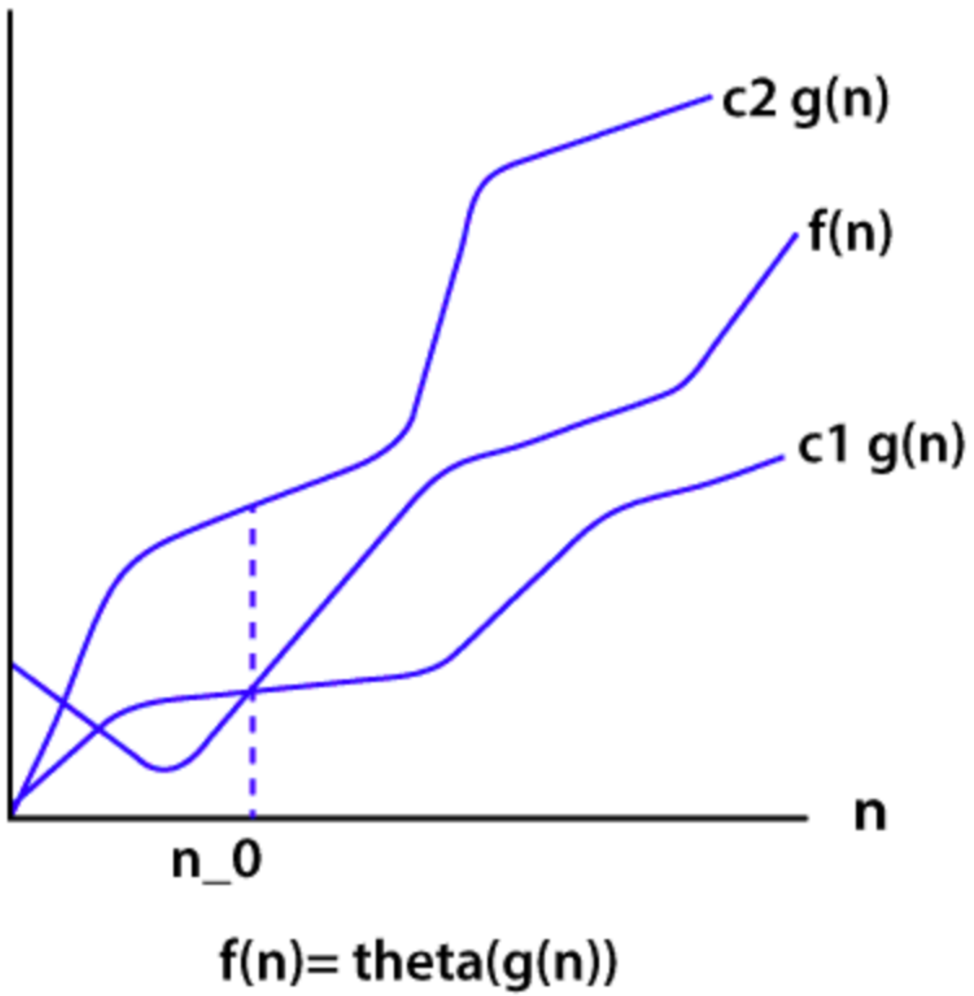
θ Notation defines the exact asymptotic behavior.

To define a behavior, it bounds functions from above and below.

A convenient way to get Theta notation of an expression is to drop low order terms and ignore leading constants.

Algorithm Interview Questions & Answers

θ Notation



Algorithm Interview Questions & Answers

Big O Notation

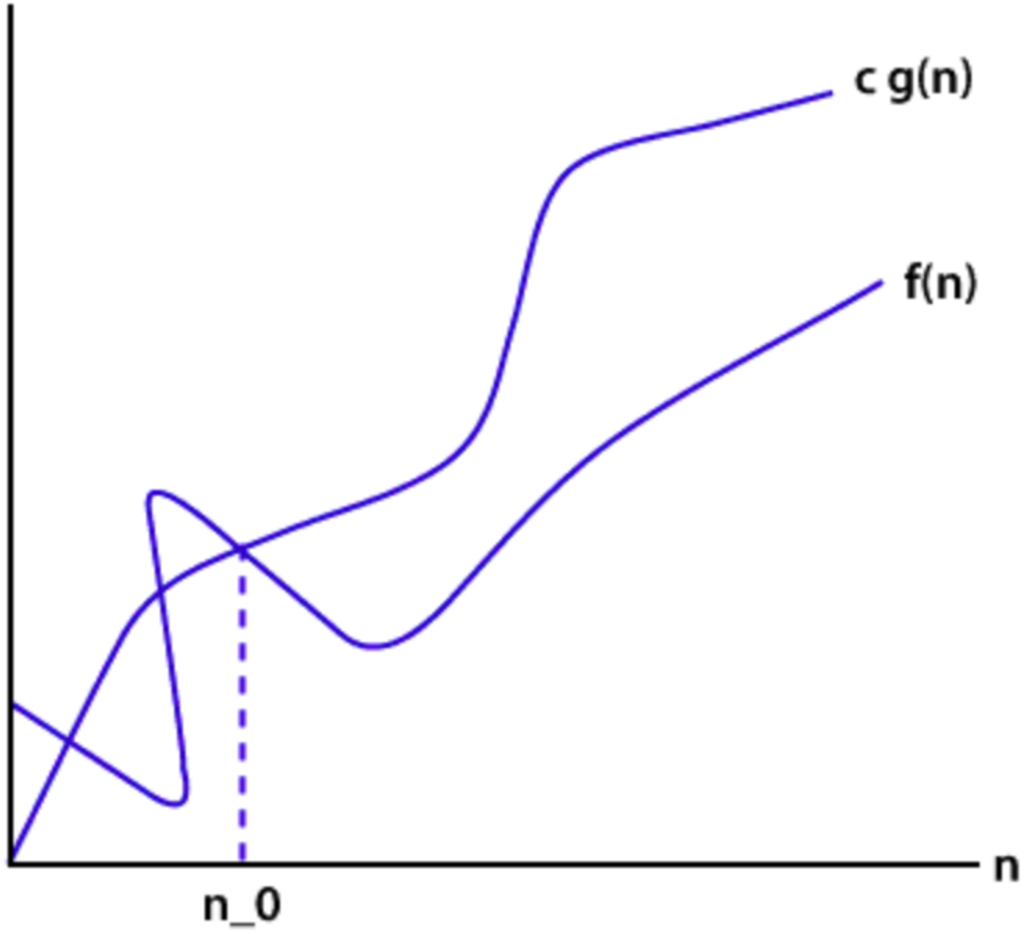
The Big O notation bounds a function from above, it defines an upper bound of an algorithm.

Let's consider the case of insertion sort; it takes linear time in the best case and quadratic time in the worst case.

The time complexity of insertion sort is $O(n^2)$. It is useful when we only have upper bound on time complexity of an algorithm.

Algorithm Interview Questions & Answers

Big O Notation



f(n)=O(g(n))

Algorithm Interview Questions & Answers

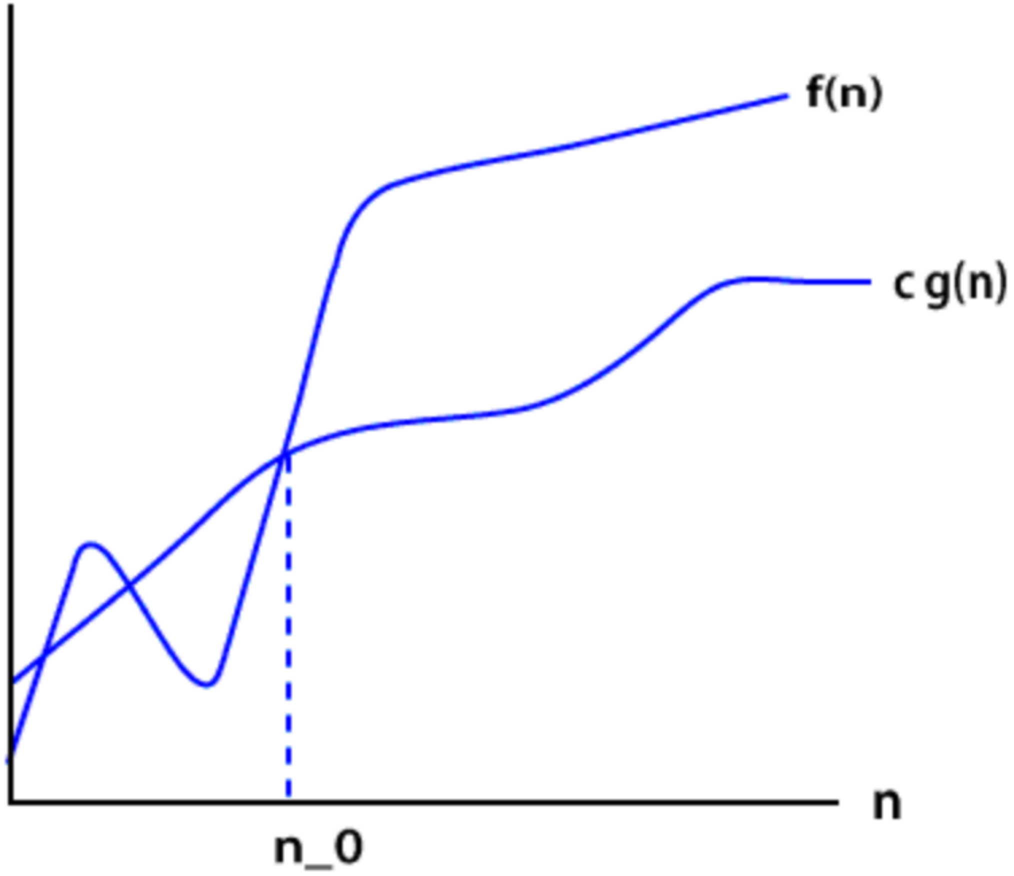
Ω Notation

Just like Big O notation provides an asymptotic upper bound, the **Ω Notation** provides an asymptotic lower bound on a function.

It is useful when we have lower bound on time complexity of an algorithm.

Algorithm Interview Questions & Answers

Ω Notation



f(n)= Omega(g(n))

Algorithm Interview Questions & Answers

Explain the Bubble Sort algorithm

Bubble sort is the simplest sorting algorithm among all sorting algorithm. It repeatedly works by swapping the adjacent elements if they are in the wrong order.

How would you use bubble sort to sort this array?

{7,2,5,3,8}

Algorithm Interview Questions & Answers

Pass1:

(**72**538) -> (27538) swap 7 and 2.

(2**75**38) -> (25738) swap 7 and 5.

(25**73**8) -> (25378) swap 7 and 3.

(253**78**) -> (25378) algorithm does not swap 7 and 8 because 7<8.

Pass2:

(2**5**378) -> (25378) algorithm does not swap 2 and 5 because 2<5.

(2**53**78) -> (23578) swap 3 and 5.

(23**57**8) -> (23578) algorithm does not swap 5 and 7 because 5<7.

(235**78**) -> (23578) algorithm does not swap 7 and 8 because 7<8.

Algorithm Interview Questions & Answers

```
void BubbleSort(int arr[], int n)
{
  for (int i = 0; i < n-1; i++)
  {
    for (int j = 0; j < n-i-1; j++)
    {
      if (arr[j] > arr[j+1])
        swap(&arr[j], &arr[j+1]);
    }
  }
}
```

Algorithm Interview Questions & Answers

n = 6

i	j
0	0,1,2,3,4
1	0,1,2,3
2	0,1,2
3	0,1
4	0

```
void BubbleSort(int arr[], int n)
{
  for (int i = 0; i < n-1; i++)
  {
    for (int j = 0; j < n-i-1; j++)
    {
    }
  }
}
```

What is the time complexity of Bubble Sort?

Algorithm Interview Questions & Answers

How to swap two integers without swapping the temporary variable?

Suppose we have two integers `i` and `j`, the value of `i=7` and `j=8` then how will you swap them without using a third variable?

```
i = i + j
j = i - j
i = i - j
```

Looks good but…

The integer will overflow if the addition is more than the maximum value of `int` as defined by `INT_MAX` and if subtraction is less than minimum value, `INT_MIN`.

Algorithm Interview Questions & Answers

The integer will overflow if the addition is more than the maximum value of `int` as defined by `INT_MAX` and if subtraction is less than minimum value, `INT_MIN`.

```
i = i + j
j = i – j
i = i - j
```

```
/* Minimum and maximum values a `signed int' can hold.  */
#  define INT_MIN (-INT_MAX - 1)
#  define INT_MAX 2147483647

Swap 2147483647 and 1

i = 2147483647 + 1 = -2147483648
j = -2147483648 – 1 = 2147483647
i = -2147483648 – 2147483649 = 1
```

This works in C but would not work in a language like Java that does not handle integer overflow.

Algorithm Interview Questions & Answers

Want to really impress?

Use the XOR method…

```
i = i ^ j;
j = i ^ j;
i = i ^ j;
```

Set `i = 7` and `j = 8`

```
i = i ^ j
i = 7 ^ 8
i = 00000111 ^ 00001000 = 00001111
i = 15

j = i ^ j
j = 15 ^ 8
j = 00001111 ^ 00001000 = 00000111
j = 7

i = i ^ j
i = 15 ^ 7
i = 00001111 ^ 00000111 = 00001000
i = 8
```

Algorithm Interview Questions & Answers

What are Divide and Conquer algorithms?

Divide and Conquer is not an algorithm; it's a pattern for an algorithm.

It is an algorithm that breaks up a large input into smaller pieces and solves the problem for each of the small pieces.

Then, all of the piecewise solutions are merged into a global solution.

This strategy is called divide and conquer.

Algorithm Interview Questions & Answers

Divide and conquer uses the following steps

Divide: This step divides the original problem into a set of subproblems.

Conquer: This step solves every subproblem individually.

Combine: This step puts together the solutions of the subproblems to get the solution to the whole problem.

Algorithm Interview Questions & Answers

Give some examples of Divide and Conquer algorithms

Merge Sort

Quick Sort

Binary Search

Algorithm Interview Questions & Answers

Explain the BFS algorithm?

BFS (Breadth First Search) is a graph traversal algorithm.

It starts traversing the graph from the root node and explores all the neighboring nodes.

It selects the nearest node and visits all the unexplored nodes.

The algorithm follows the same procedure for each of the closest nodes until it reaches the goal state.

Algorithm Interview Questions & Answers

What is Dijkstra's shortest path algorithm?

Dijkstra's algorithm is an algorithm for finding the shortest path from a starting node to the target node in a weighted graph. The algorithm makes a tree of shortest paths from the starting vertex and source vertex to all other nodes in the graph.

Suppose you want to go from home to school using the shortest possible way. You know some roads are heavily congested and using those routes will take more time (meaning these edges have a larger weight).

Dijkstra's algorithm can be used to find the shortest path.

Algorithm Interview Questions & Answers

What are Greedy algorithms?

A greedy algorithm is an algorithmic strategy which is made for the best optimal choice at each sub stage with the goal of this, eventually leading to a globally optimum solution. This means that the algorithm chooses the best solution at the moment without regard for consequences.

In other words, an algorithm that always takes the best immediate, or local, solution while finding an answer.

Greedy algorithms find the overall, ideal solution for some idealistic problems, but may discover less-than-ideal solutions for some instances of other problems.

Algorithm Interview Questions & Answers

List some Greedy algorithms?

Prim

Kruskal

Dijkstra

Algorithm Interview Questions & Answers

What is a linear search?

Linear search is used on a group of items. It relies on the technique of traversing a list from start to end by visiting properties of all the elements that are found on the way.

Step1: Traverse the array using **for loop**.

Step2: In every iteration, compare the target value with the current value of the array

Step3: If the values match, return the current index of the array

Step4: If the values do not match, shift on to the next array element.

Step5: If no match is found, return -1

Algorithm Interview Questions & Answers

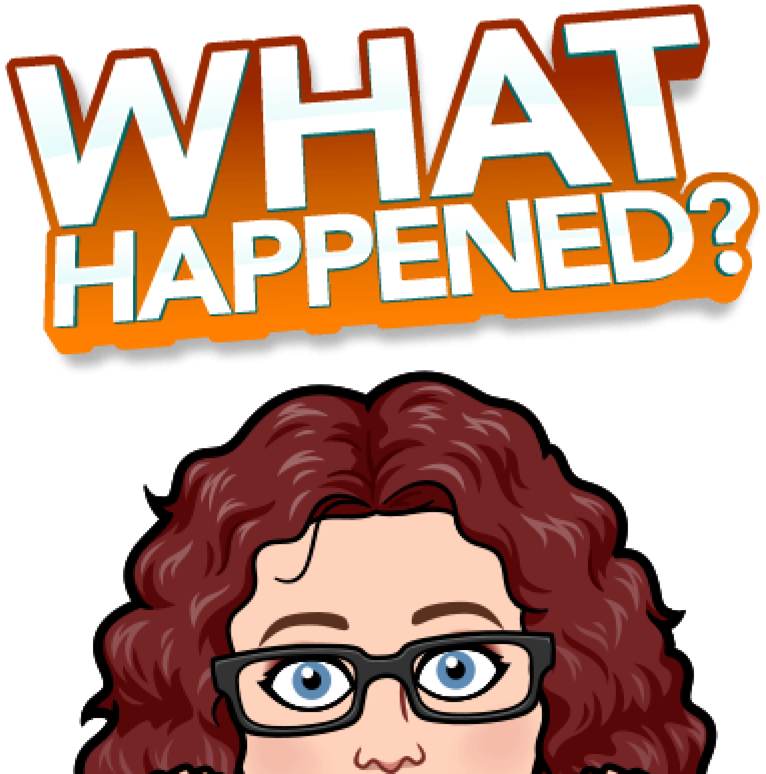
What is a Binary Search Tree?

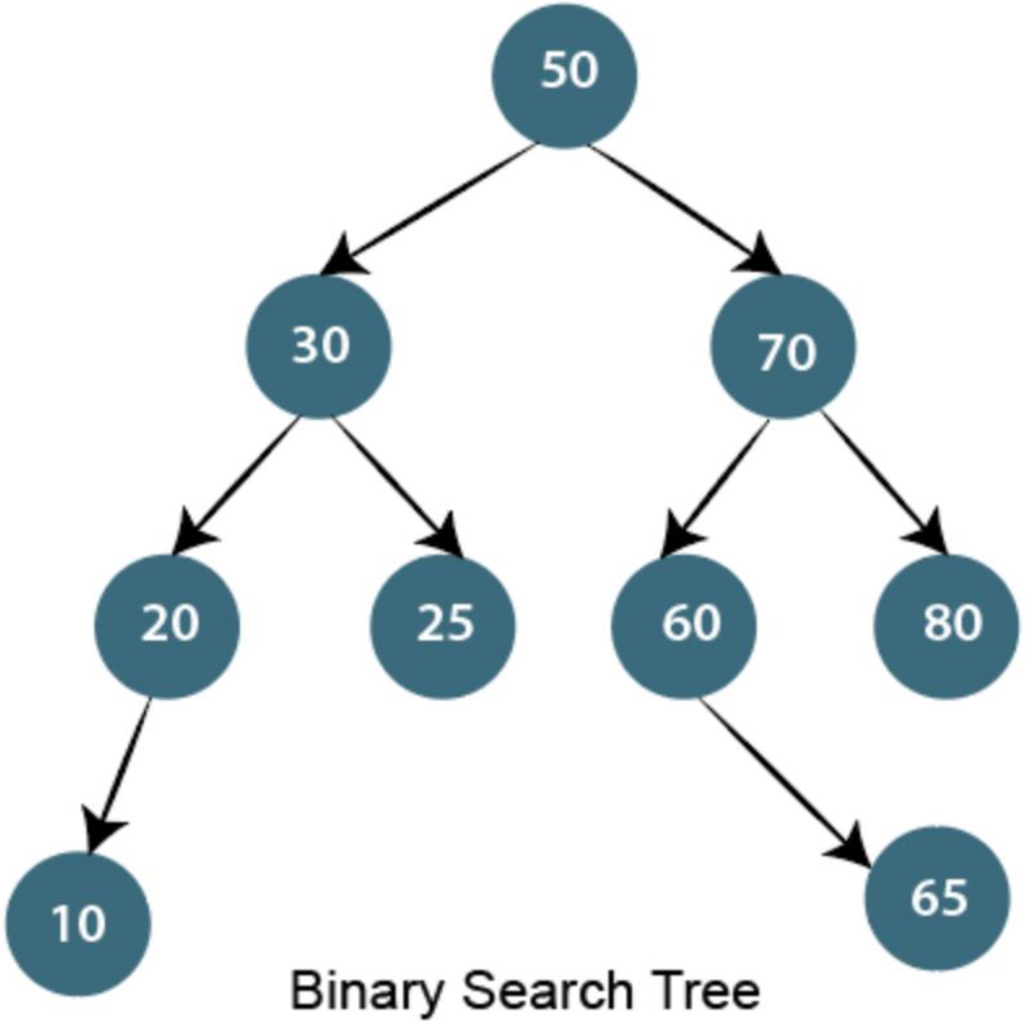
The binary search tree is a special type of data structure which has the following properties.

- Nodes which are less than root will be in the left subtree.
- Nodes which are greater than root will be right subtree.
- A binary search tree should not have duplicate nodes.
- Both sides subtree (left and right) also should be a binary search tree.

Algorithm Interview Questions & Answers

50
30
70
20
25
60
80
10
65





Algorithm Interview Questions & Answers

Write an algorithm to insert a node in the Binary search tree

Compare the node to be inserted with the root node and traverse left (if smaller) or right (if greater) according to the value of the node to be inserted.

Algorithm

- Set root node as the current node
- If the node to be inserted < root
    - If it has left child, then traverse left
    - If it does not have left child, insert node here
- If the node to be inserted > root
    - If it has the right child, traverse right
    - If it does not have the right child, insert node here.

Algorithm Interview Questions & Answers

Count the leaf nodes of the binary tree

Algorithm

Traverse the tree (inorder, preorder or post order)

If a node is a leaf (both right and left pointers are NULL),
then increment leaf counter

Algorithm Interview Questions & Answers

What is the difference between the Singly Linked List and Doubly Linked List?

This is a traditional interview question on the data structure. The major difference between the singly linked list and the doubly linked list is the ability to traverse.

You cannot traverse back in a singly linked list because in it a node only points towards the next node and there is no pointer to the previous node.

On the other hand, the doubly linked list allows you to navigate in both directions in any linked list because it maintains two pointers towards the next and previous node.

Algorithm Interview Questions & Answers

What is a hash algorithm and how is it used?

You will want to get comfortable answering this question because hash algorithms are popular now due to their use in cryptography.

A hash algorithm refers to a hash function, which takes a string and converts it to a fixed length regardless of how long it was to begin with.

You can use it for a wide range of applications, from cryptocurrency to passwords and a range of other validation tools.

Algorithm Interview Questions & Answers

What are the three laws that govern a recursive algorithm?

These kinds of algorithm interview questions may come as follow-ups for the "What is a recursive algorithm?" question.

A recursive algorithm needs to follow these laws:

It has to have a base case.

It has to call itself.

It needs to change its state and shift towards the base case.

Algorithm Interview Questions & Answers

What Are the Different Types of Data Structures?

A collection of data values stored sequentially	Arrays
Last-in-first-out (LIFO) data structures where the element placed last is accessed first.	Stacks
A first-in-first-out data structure.	Queues
A collection of data values stored in a linear order and connected to each other	Linked lists
A data structure in which data values are placed in nodes connected by edges	Graphs
Similar to a linked list, but with data values linked in a hierarchical fashion	Trees
A binary tree data structure wherein parent data values can be compared to child data values	Heaps
A table where each value is assigned a key and then stored, making accessing individual values easy.	Hash table