AlgoTutor

# 15 Important Algorithm

## Every Programmer Should Know

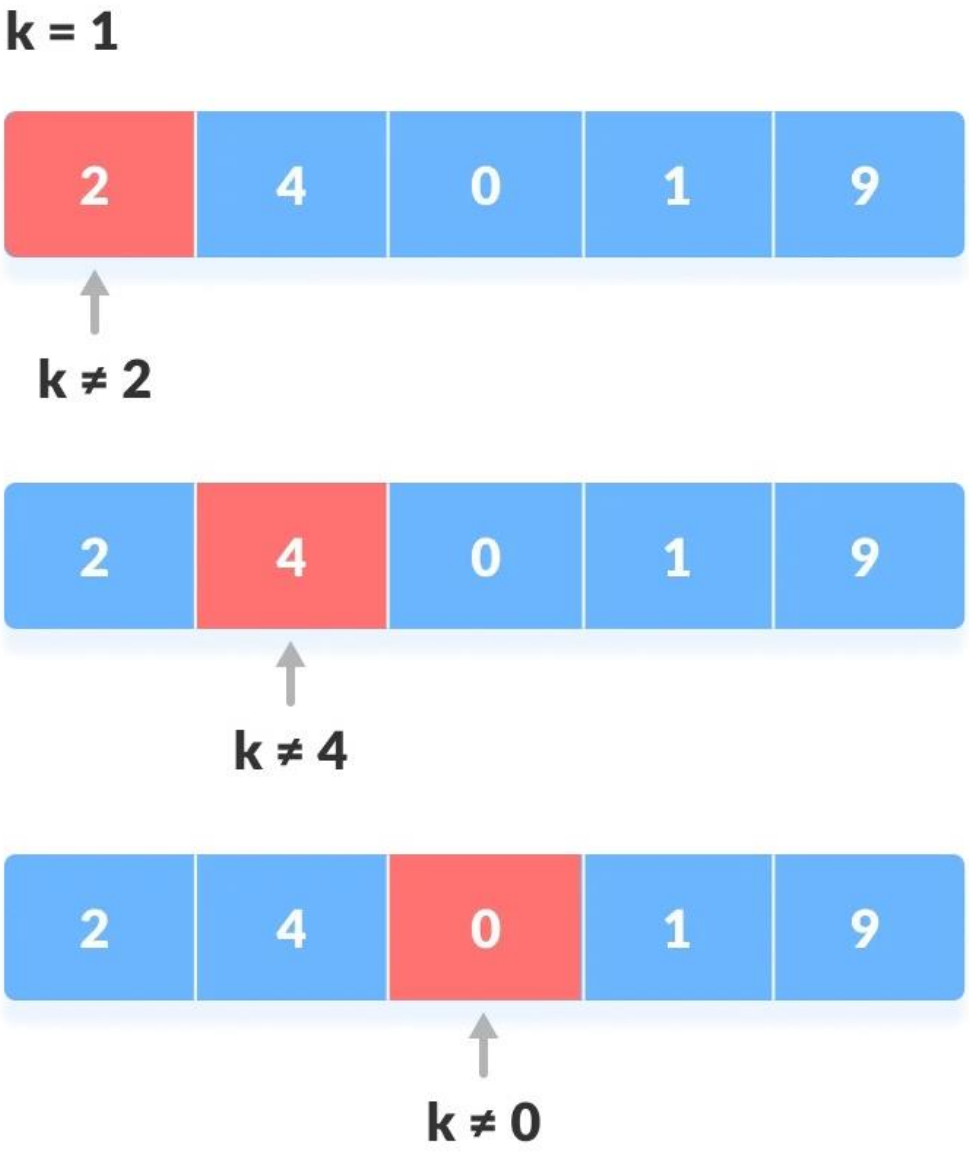
+91-7260058093
info@algotutor.io
www.algotutor.io

# Searching Algorithms

## Linear Search:

Linear search is a simple searching algorithm that sequentially checks each element in a list or array until it finds the target value or reaches the end of the list. It is also known as a sequential search.

**Time Complexity: O(n)**



## Sample Problems:

- **Kth Missing Positive Number**

  Given an array arr of positive integers sorted in a strictly increasing order, and an integer k.

  Return the kth positive integer that is missing from this array.

  **Input: arr** = [2,3,4,7,11], **k** = 5
  **Output:** 9

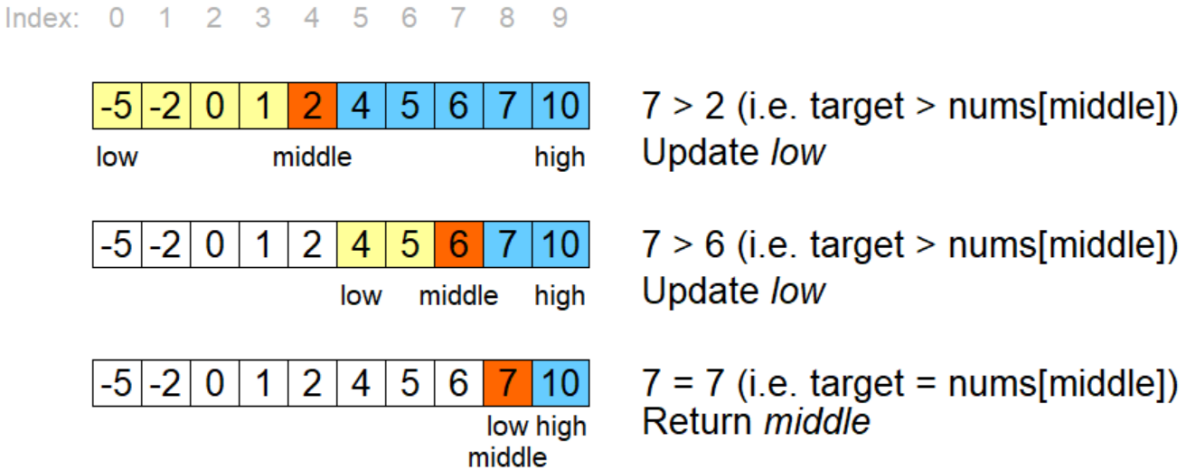
- **Searching a number**

  Given an array Arr of N elements and a integer K. Your task is to return the position of first occurence of K in the given array.

  **Input: N =** 5**, K =** 16
  **Arr [ ] =** {9, 7, 2, 16, 4}
  **Output:** 4

# Binary Search:

Binary search is an efficient searching algorithm that works on sorted lists or arrays. It follows a divide-and-conquer approach, repeatedly dividing the search space in half until the target element is found or determined to be absent.




## Time Complexity: $O(\log_2 n)$

## Sample Problems:

- **Binary Search**

  Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

  You must write an algorithm with O(log n) runtime complexity.

  **Input: nums =** [-1,0,3,5,9,12], **target =** 9
  **Output:** 4

- **Cake Distribution Problem**

  Geek is organizing a birthday party, so his friends brought a cake for him. The cake consists of N chunks, whose individual sweetness is represented by the sweetness array. Now at the time of distribution, Geek cuts the cake into K + 1 pieces to distribute among his K friends. One piece he took for himself. Each piece consists of some consecutive chunks. He is very kind, so he left the piece with minimum sweetness for him.

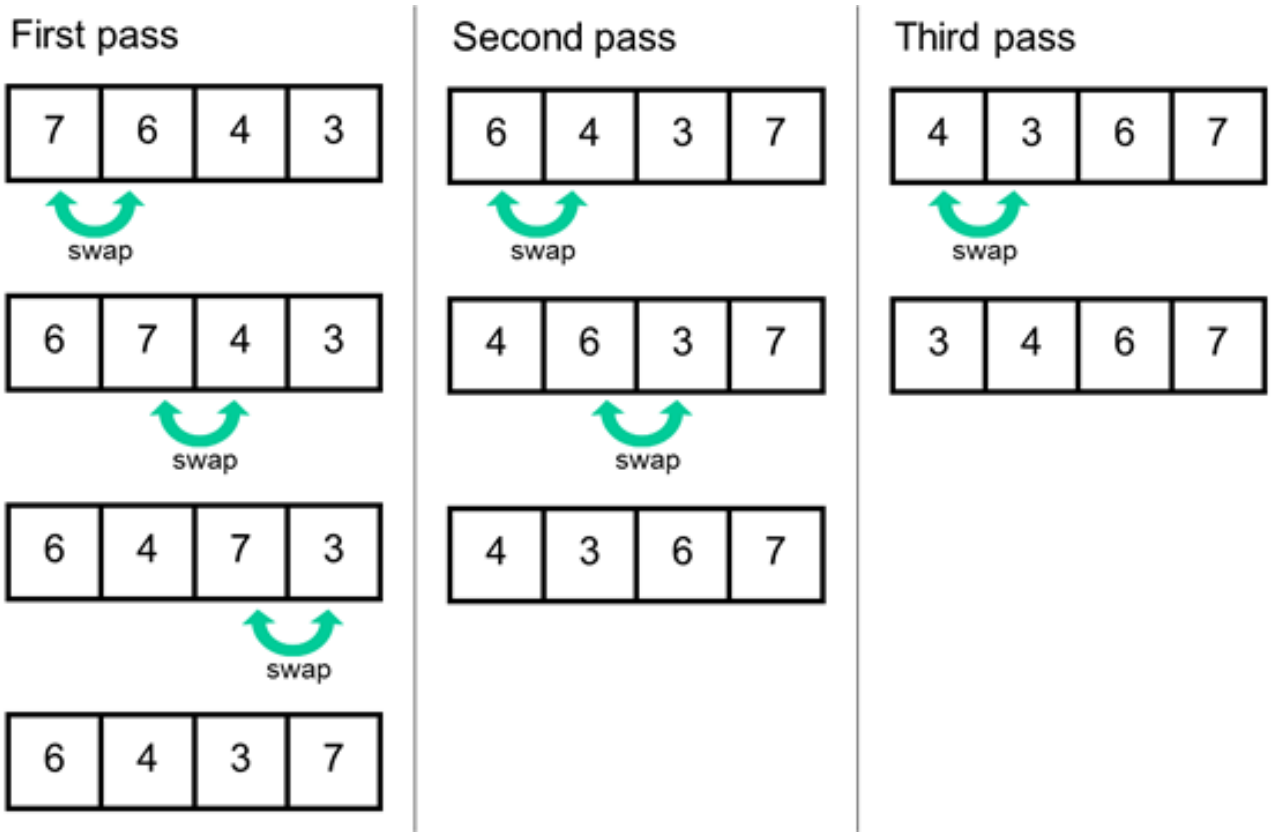
  You need to find the maximum sweetness that the Geek can get if he distributes the cake optimally.

  **Input: N = 6, K =** 2
  **sweetness[ ] =** {6, 3, 2, 8, 7, 5}
  **Output:** 9

# Sorting Algorithms

## Bubble Sort Algorithms:

Bubble sort is a simple sorting algorithm that repeatedly steps through the list or array, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm gets its name from the way smaller elements "bubble" to the top of the list with each iteration.




## Time Complexity: $O(n^2)$

## Sample Problems:

- **Sort Colors**

  Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

  We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

  You must solve this problem without using the library's sort function.

  **Input: nums =** [2,0,2,1,1,0]
  **Output:** [0,0,1,1,2,2]

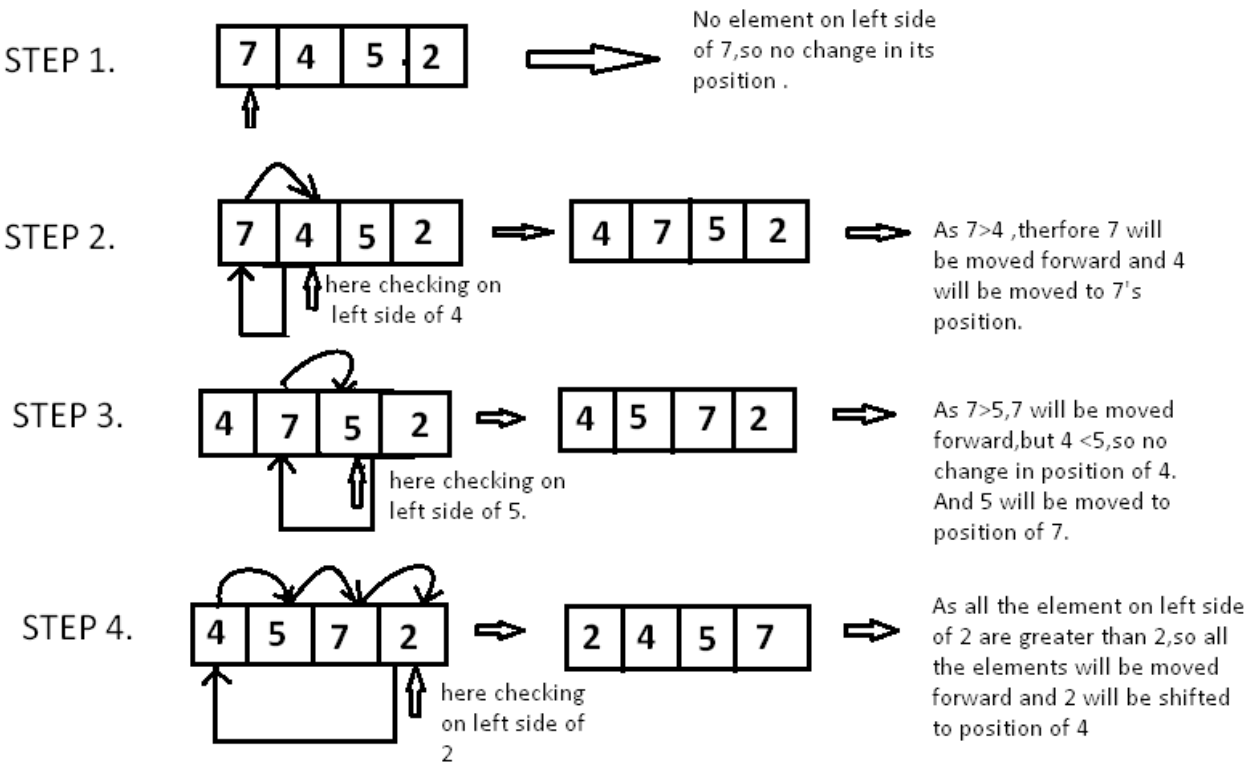
- **Bubble Sort**

  Given an Integer N and a list arr. Sort the array using bubble sort algorithm.

  **Input: N =** 5
  **arr[ ] =** {4, 1, 3, 9, 7}
  **Output:** 1 3 4 7 9

# Insertion Sort Algorithms:

Insertion sort is a simple sorting algorithm that builds the final sorted array one element at a time. It iterates through the input list or array and gradually expands a sorted portion of the list by inserting each new element in its proper place.




## Time Complexity: $O(n^2)$

## Sample Problems:

- **Insertion Sort**

  The task is to complete the **insert()** function which is used to implement Insertion Sort.

  **Input:**
  N = 5, **arr[ ] =** { 4, 1, 3, 9, 7}
  **Output:** 1 3 4 7 9

- **Insertion Sort ||**

  The first line contains one space separated integer N denoting the number of elements.

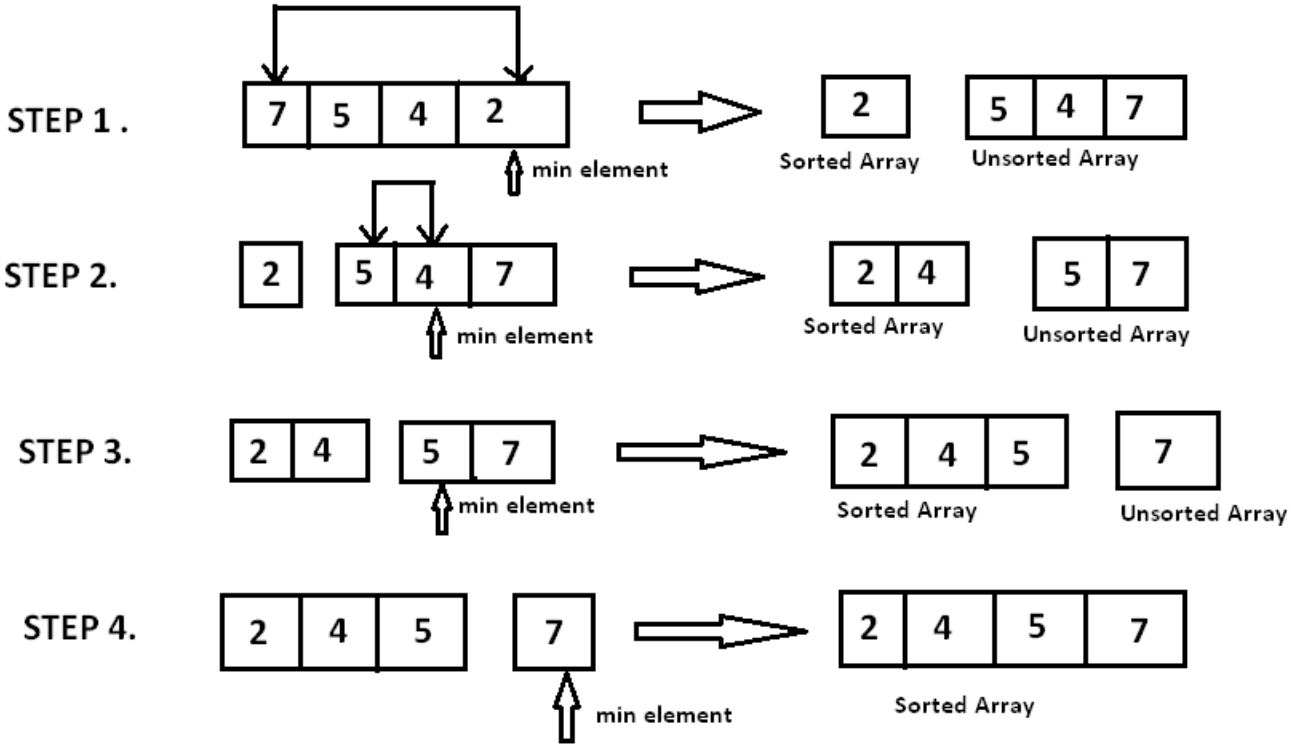
  The Second line contains N space separated integers denoting the elements of the array.

  You need to complete insertionSort function which contains a array of size n and print the final sorted array in this function only.

  **Input: 7 6 8 5 4 9**
  **Output: 4 5 6 7 8 9**

# Selection Sort Algorithms:

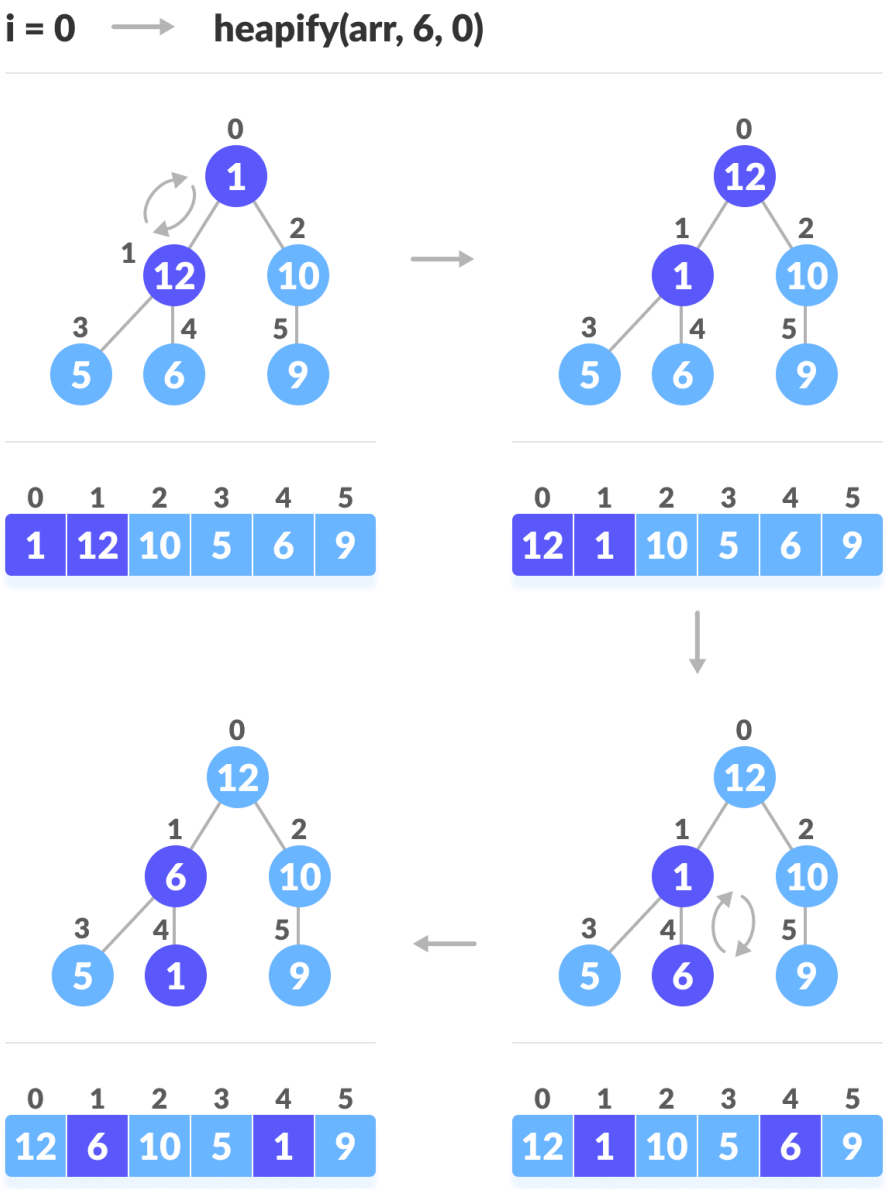
Selection sort is a simple sorting algorithm that divides the input list into two parts: the sorted portion at the beginning and the unsorted portion at the end. It repeatedly selects the smallest element from the unsorted portion and swaps it with the element at the beginning of the unsorted portion, thus expanding the sorted portion by one element.




## Time Complexity: $O(n^2)$

# Heap Sort Algorithms:

Heap sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort elements. It involves building a heap from the input list and repeatedly extracting the maximum (for ascending order) or minimum (for descending order) element from the heap and placing it at the end of the sorted portion of the list.




## Time Complexity: O(n*log n)

## Sample Problems:

- **Heap Sort**

  Given an array of size N. The task is to sort the array elements by completing functions heapify() and buildHeap() which are used to implement Heap Sort.

  **Input:**
  **N =** 5**, arr [ ] =** {4,1,3,9,7}
  **Output:** 1 3 4 7 9

- **Heap Sort** ||

  Given an array of size N. The task is to sort the array elements by implementing Heap Sort.

  - Hint: Make two functions heapify() and heapSort().
  - The passed array needs to be sorted

  **Input Format**

  The first line contains an integer N, the size of the array.

  The second line contains N spaced integers, the elements of the array.

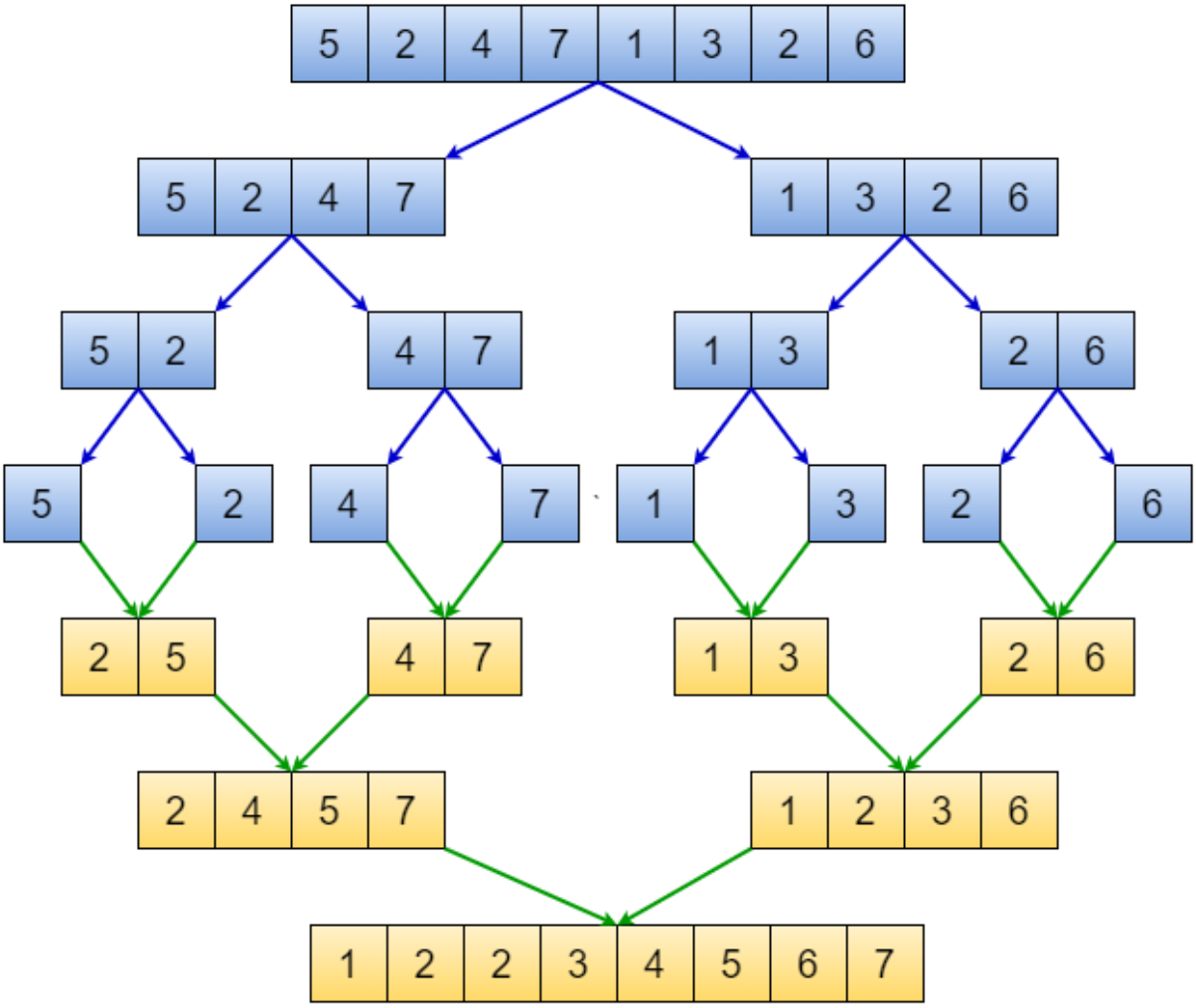
  **Output Format**

  Print the sorted array.

  **Input: 4 1 3 9 7**
  **Output: 1 3 4 7 9**

# Merge Sort Algorithms:

Merge sort is a recursive sorting algorithm that follows the divide-and-conquer principle. It divides the input list or array into smaller subproblems, sorts them independently, and then merges them to obtain a sorted result.




## Time Complexity: O(n*log n)

## Sample Problems:

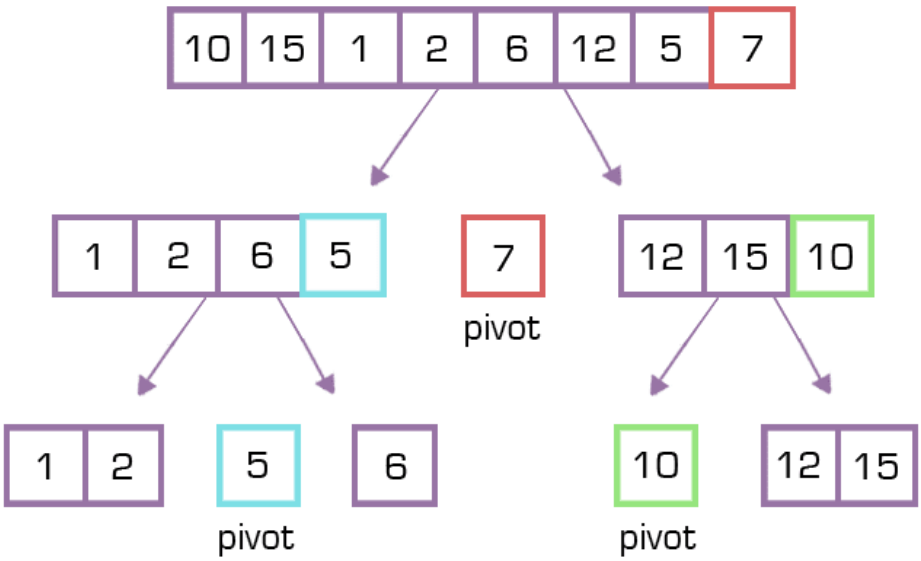
- **Merge Sort**

  Given an array arr[ ], its starting position l and its ending position r. Sort the array using merge sort algorithm.

  **Input: N =** 5**, arr[ ] =** {4 1 3 9 7}
  **Output:** 1 3 4 7 9

# Quick Sort Algorithms:

Quick sort is a highly efficient, comparison-based sorting algorithm that follows the divide-and-conquer principle. It works by partitioning the input list or array into two subarrays based on a chosen pivot element, and recursively sorting the subarrays.




## Time Complexity: O(n*log n)

## Sample Problems:

- **Quick Sort**

  Quick Sort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot.

  Given an array arr[], its starting position is low (the index of the array) and its ending position is high(the index of the array).

  **Input: N =** 5**, arr[ ] =** { 4, 1, 3, 9, 7}
  **Output:**1 3 4 7 9

AlgoTutor

# Why Choose AlgoTutor?

- 100 % PLACEMENT ASSISTANCE
- 1:1 PERSONAL MENTORSHIP FROM INDUSTRY EXPERTS
- 100 % SUCCESS RATE
- 23 LPA(AVG.)CTC
- 200+ SUCCESSFUL ALUMNI
- 147%(AVG.) SALARY HIKE
- LEARN FROM SCRATCH
- CAREER SERVICES

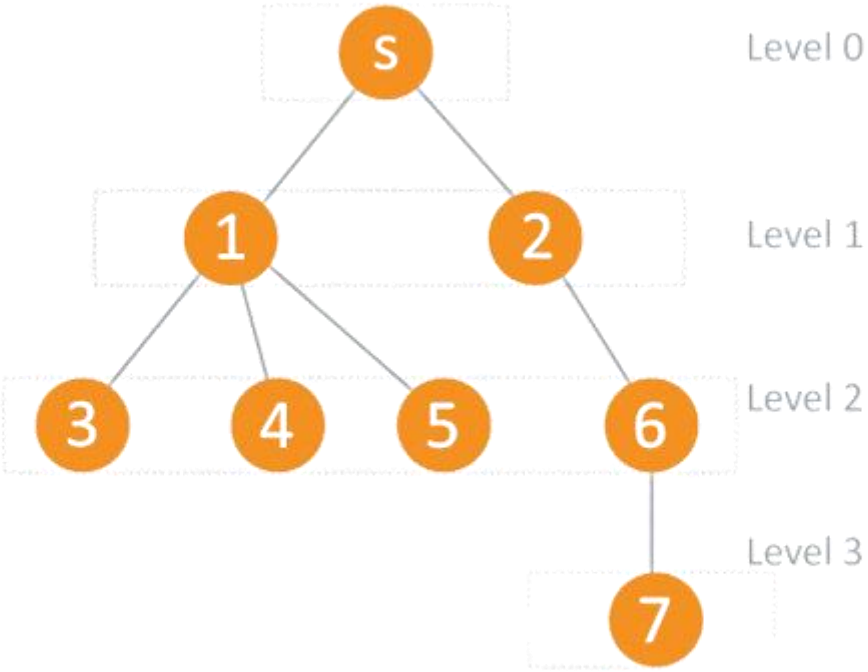
**For More Information Contact Us:**

+91-7260058093 || info@algotutor.io || www.algotutor.io

# Graph Algorithms

## Breadth-first search:

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level



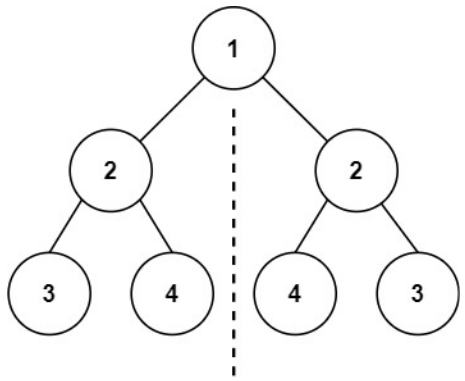

## Time Complexity: O(V + E)

## Sample Problems:

- **Symmetric Tree**

  Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

  **Example 1:**




  **Input: root =** [1,2,2,3,4,4,3]
  **Output:** true

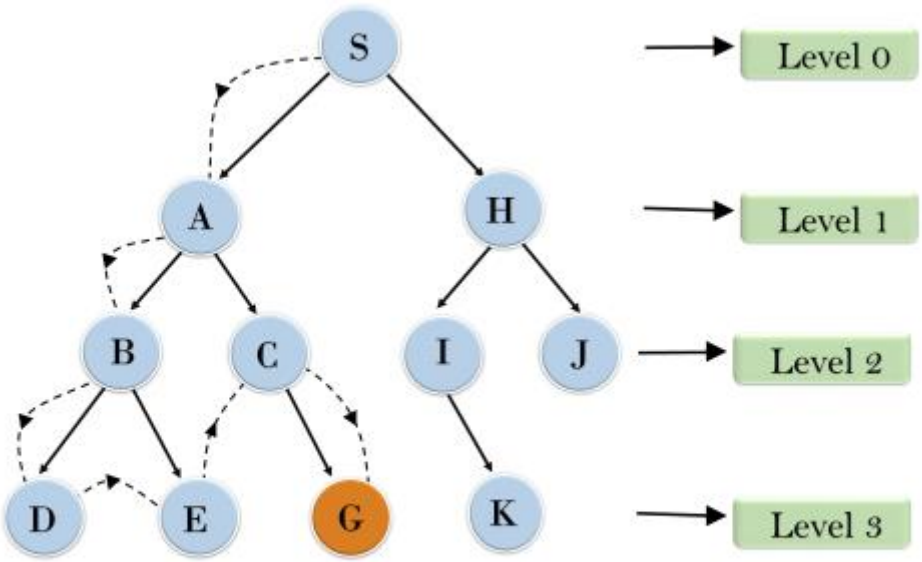
- **Shortest Prime Path**

  You are given two four digit prime numbers Num1 and Num2. Find the distance of the shortest path from Num1 to Num2 that can be attained by altering only one single digit at a time. Every number obtained after changing a digit should be a four digit prime number with no leading zeros.

  **Input:**
  **Num1 =** 1033
  **Num2 =** 8179
  **Output:** 6

## Depth-first search:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking.




## Time Complexity: O(V + E)

## Sample Problems:

- **Binary Tree Inorder Traversal**

  Given the root of a binary tree, return the inorder traversal of its nodes' values.

  **Input: root =** [1,null,2,3]
  **Output:** [1,3,2]

- **X Total Shapes**

  Given  a grid of n*m consisting of O's and X's. The task is to find the number of 'X' total shapes.

  **Note**: 'X' shape consists of one or more adjacent X's (diagonals not included).

  **Input: grid =** {{X,O,X},{O,X,O},{X,X,X}}
  **Output:** 3

- **Select Nodes**

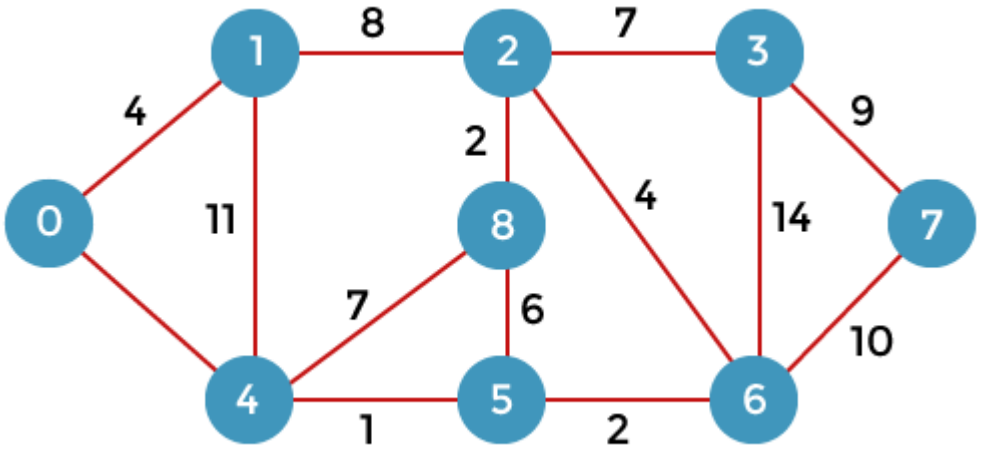
  Given N nodes of a tree and a list of edges. Find the minimum number of nodes to be selected to light up all the edges of the tree.

  An edge lights up when at least one node at the end of the edge is selected.

  **Input: N =** 6
  **Edges [ ] =** {(1,2), (1,3), (2,4), (3,5), (3,6)}
  **Output:** 2

# Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a weighted graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants




## Time Complexity: O ( V + E log V ) .

## Sample Problems:

- **Implementing Dijkstra Algorithm**

  Given a weighted, undirected and connected graph of V vertices and an adjacency list adj where adj[i] is a list of lists containing two integers where the first integer of each list j denotes there is edge between i and j , second integers corresponds to the weight of that edge . You are given the source vertex S and You to Find the shortest distance of all the vertex's from the source vertex S. You have to return a list of integers denoting shortest distance between each node and Source vertex S.

  **Input: V** = 2

  **adj [ ]** = **{{{1, 9}}, {{0, 9}}}, S** = 0

  **Output:** 0 9

- **Path with Maximum Probability**

  You are given an undirected weighted graph of n nodes (0-indexed), represented by an edge list where edges[i] = [a, b] is an undirected edge connecting the nodes a and b with a probability of success of traversing that edge succProb[i].

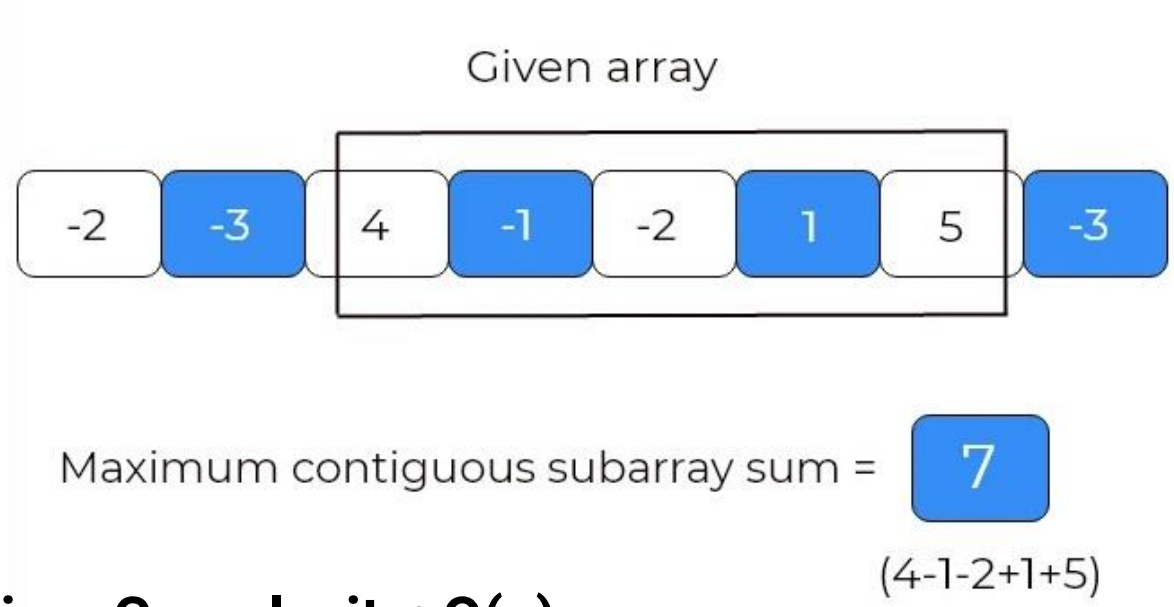
  Given two nodes start and end, find the path with the maximum probability of success to go from start to end and return its success probability.

  **Input: n =** 3**, edges =** [[0,1],[1,2],[0,2]], **succProb =** [0.5,0.5,0.2], **start =** 0**, end =** 2
  **Output:** 0.25000

# Kadane’s Algorithm

## Kadane’s Algorithm:

Kadane's algorithm is an efficient algorithm used to find the maximum sum of a contiguous subarray within a given array of numbers. It is often used to solve the maximum subarray problem.




## Time Complexity: O(n)

## Sample Problems:

- **Maximum Subarray**

  Given an integer array nums, find the subarray with the largest sum, and return its sum.

  **Input: nums** = [-2,1,-3,4,-1,2,1,-5,4]
  **Output:** 6

- **Save Your Life**

  Given a string w, integer array b, character array x and an integer n. n is the size of array b and array x. Find a substring which has the sum of the ASCII values of its every character, as maximum. ASCII values of some characters are redefined.

  **Input:**
  **W =** abcde
  **N =** 1
  **X[ ] =** { 'c' }
  **B[ ] =** { -1000 }
  **Output:** de

- **Max Sum**

  Find max sum contiguous subarray.

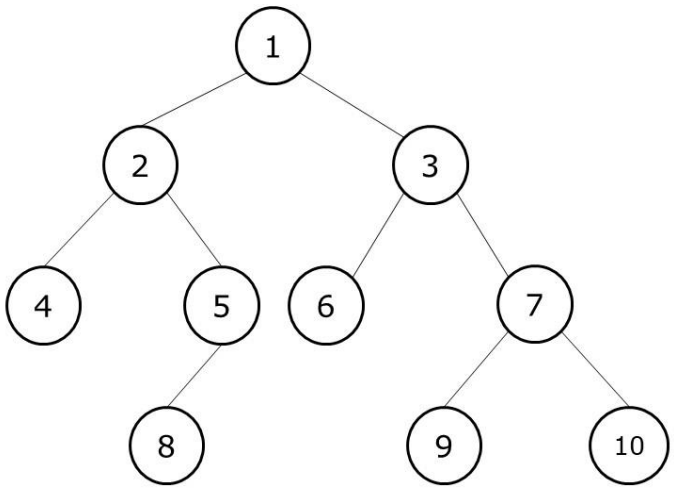
  **Input:** [-3, -4, 5, -1, 2, -4, 6, -1]
  **Output:** 8

# Tree Algorithm

## Inorder Traversal:

- In an inorder traversal, the left subtree is visited first, followed by the root node, and then the right subtree.
- The inorder traversal visits the nodes of a binary tree in ascending order when applied to a binary search tree (BST).
- The pseudocode for inorder traversal is as follows:

```
inorderTraversal(node):
    if node is null:
        return
    inorderTraversal(node.left)
    visit(node)
    inorderTraversal(node.right)
```




Inorder Traversal:
[left,root,right]

| 4 | 2 | 8 | 5 | 1 | 6 | 3 | 9 | 7 | 10 |
|---|---|---|---|---|---|---|---|---|---|

## Time Complexity: O(n)

## Sample Problems:

- **Inorder Traversal**

  Given a Binary Tree, find the In-Order Traversal of it.

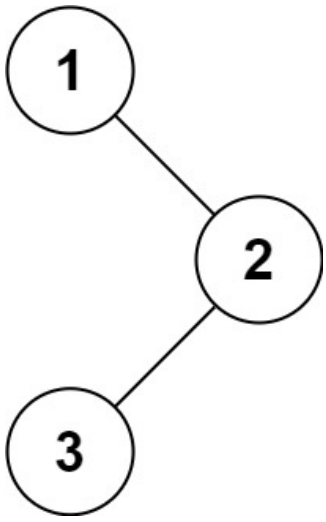
  **Input:**


  **Output:** 3 1 2

- **Inorder Traversal ||**

  Given the root of a binary tree, return the inorder traversal of its nodes' values.

  **Input:**



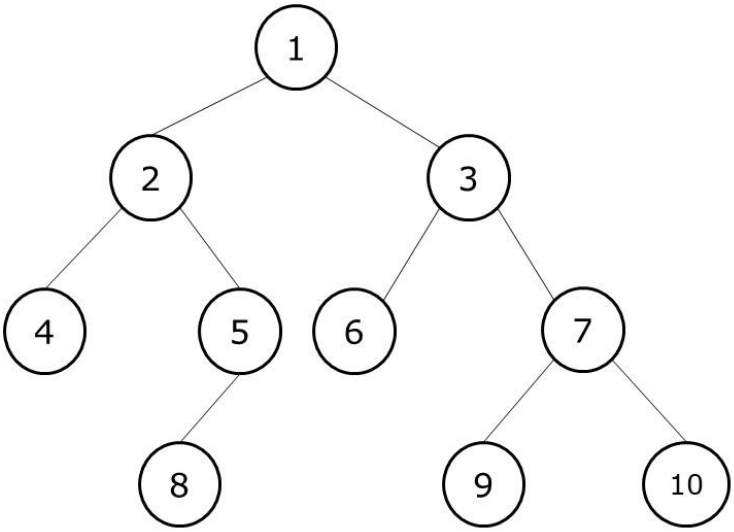



  **Input: root =** [1,null,2,3]
  **Output:** [1,3,2]

# Preorder Traversal:

- In a preorder traversal, the root node is visited first, followed by the left subtree, and then the right subtree.
- The preorder traversal is useful for creating a copy of a tree, as the nodes are visited in the order they appear in the original tree.
- The pseudocode for preorder traversal is as follows:

```
preorderTraversal(node):
    if node is null:
        return
    visit(node)
    preorderTraversal(node.left)
    preorderTraversal(node.right)
```




Preorder Traversal:
[root, left, right]

| 1 | 2 | 4 | 5 | 8 | 3 | 6 | 7 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|

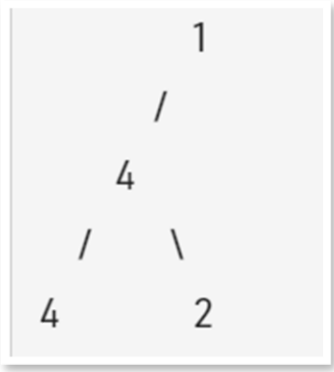
## Time Complexity: O(n)

## Sample Problems:

- **Preorder Traversal**

  Given a binary tree, find its preorder traversal.

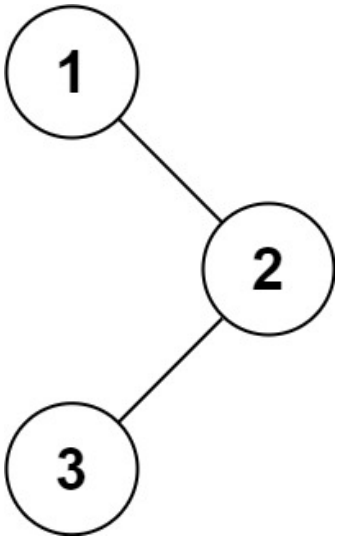
  **Input:**




  **Output:** 1 4 4 2

- **Preorder Traversal** ||

  Given the root of a binary tree, return the preorder traversal of its nodes' values.

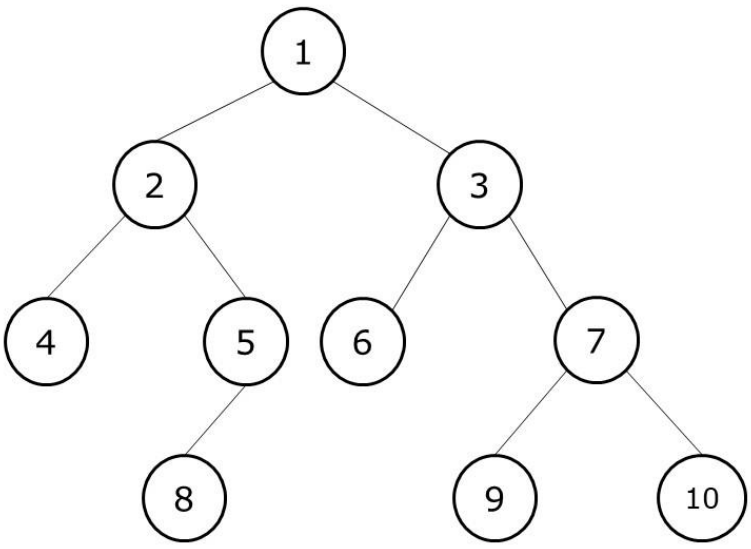



  **Input: root =** [1,null,2,3]
  **Output:** [1,2,3]

# Postorder Traversal:

- In a postorder traversal, the left subtree is visited first, followed by the right subtree, and finally the root node.
- The postorder traversal is commonly used to delete nodes from a tree since it ensures that a node's children are deleted before the node itself.
- The pseudocode for postorder traversal is as follows:

```
postorderTraversal(node):
    if node is null:
        return
    postorderTraversal(node.left)
    postorderTraversal(node.right)
    visit(node)
```




Postorder Traversal:
[left, right, root]

| 4 | 8 | 5 | 2 | 6 | 9 | 10 | 7 | 3 | 1 |
|---|---|---|---|---|---|---|---|---|---|

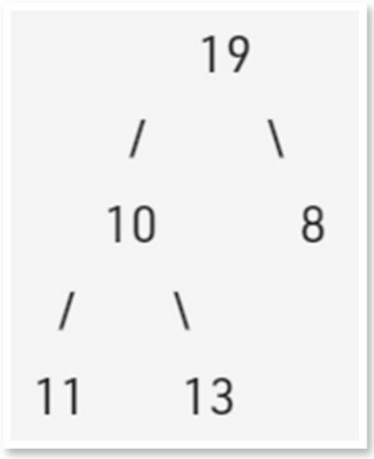
## Time Complexity: O(n)

## Sample Problems:

- **Postorder Traversal**

  Given a binary tree, find the Postorder Traversal of it.

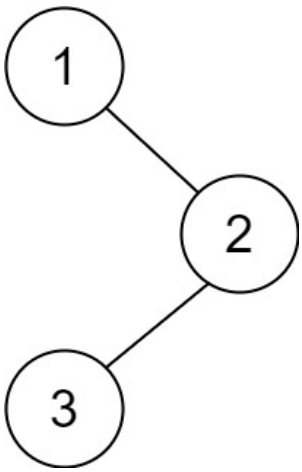
  **Input:**




  **Output:** 11 13 10 8 19

- **Postorder Traversal** ||

  Given the root of a binary tree, return the postorder traversal of its nodes' values.




  **Input: root =** [1,null,2,3]
  **Output:** [3,2,1]

AlgoTutor

# Want to up your Skill?
# Join our Popular courses

**DSA WITH SYSTEM DESIGN (HLD + LLD)**

**DSA & SYSTEM DESIGN WITH FULL STACK**

**ADVANCED DATA SCIENCE & MACHINE LEARNING**

**MERN FULL STACK WEB DEVELOPMENT**

**For More Information Contact Us:**

+91-7260058093 || info@algotutor.io || www.algotutor.io

AlgoTutor

# Why Choose AlgoTutor?

- 100 % PLACEMENT ASSISTANCE
- 1:1 PERSONAL MENTORSHIP FROM INDUSTRY EXPERTS
- 100 % SUCCESS RATE
- 23 LPA(AVG.)CTC
- 200+ SUCCESSFUL ALUMNI
- 147%(AVG.) SALARY HIKE
- LEARN FROM SCRATCH
- CAREER SERVICES

**For More Information Contact Us:**

+91-7260058093 || info@algotutor.io || www.algotutor.io