ScholarHat

Swipe

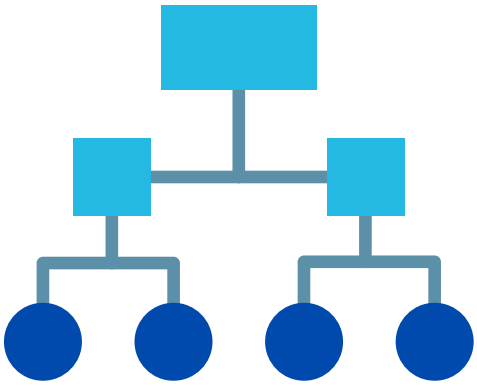
Data Structures & Algorithms (DSA)

ROADMAP (LEVEL-1)

For Beginners

2024 EDITION

LEVEL UP YOUR CAREER



AMIT KUMAR GHOSH
MENTOR AND VICE PRESIDENT AT SCHOLARHAT

DSA Career Scope

Why learn DSA?

DSA is a fundamental skill for careers in software development, data science, Machine learning, and many more. These stats back the statement.

~70% Service-based companies demands DSA Now	**~10X** Efficiency of writing code will increase.	**∞** Unlock unlimited career opportunities
INR 3-10 LPA Salary In Service based Companies	**~** Language independent	**Logic Building** Help in building logic and write clean Code.

World Top Companies Using DSA

1 Fundamentals of Java Programming

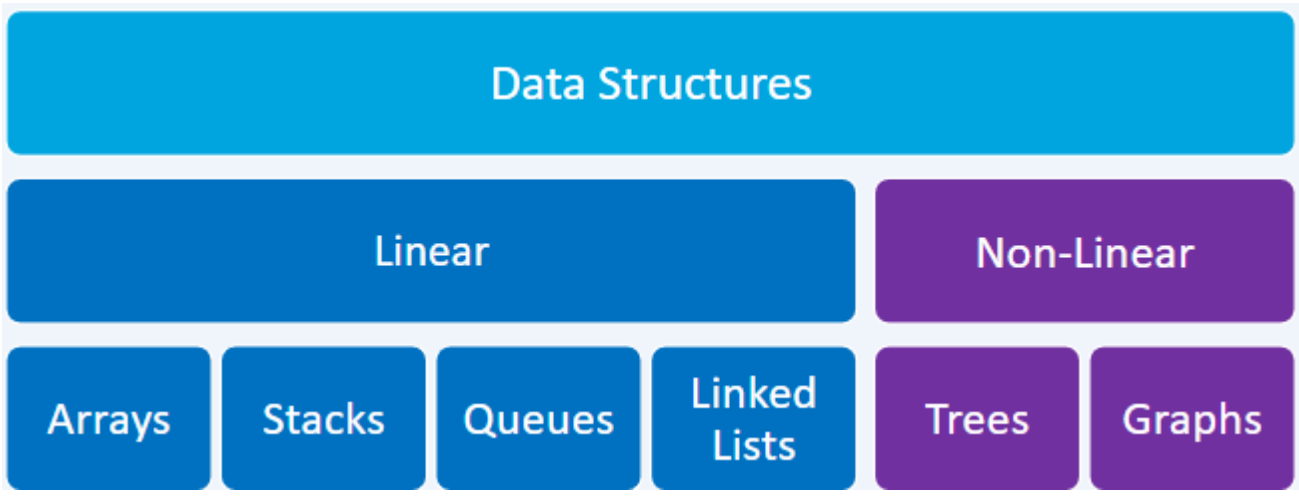
- **Variables and Data Types**
  - Learn about primitive data types (int, float, double, char, boolean) and how to declare variables.
- **Operators**
  - Understand arithmetic, relational, logical, and bitwise operators.
- **Conditional Statements and Loops**
  - Learn about if-else statements, switch-case, and loops (for, while, do-while).
- **Arrays and Strings**
  - Learn to declare and initialize arrays and strings.
  - Understand different operations performed on Arrays and Strings.
- **Functions and Methods**
  - Understand functions and their role in programming.
  - Learn how to pass parameters to functions.



2 Data Structures Fundamentals

Understand the Model-View-Controller architectural pattern.

1. **Introduction to Data Structures:**
   - A way of organizing and storing the data in memory.
   - Understand types of Data Structures: **Linear** and **Non-Linear**.
2. **Importance of Data Structures in Programming:**
   - Before starting up with DSA, you should be familiar with the importance and advantages of Data Structures.



3 Abstract Data Type (ADT)

Abstract Data Types (ADTs) represent a high-level view of data structures, focusing on the operations that can be performed on the data rather than the implementation details.

Here's a roadmap to guide you through learning about Abstract Data Types:

- Grasp the concept of ADTs as a high-level description of data and operations.
- Differentiate between ADTs and the actual data structures implementing them.
- Explore common ADTs like Lists, Stack, Queues through some examples.

4 Big-O Notation

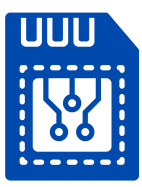
Understanding Big-O notation is crucial for analyzing the efficiency of algorithms in Data Structure and Algorithm (DSA) development. Here's a Roadmap:

1. **Introduction to Space and Time Complexity:**
   - Understand time complexity as an amount of time an algorithm takes to complete.
   - Learn about space complexity as an amount of memory space an algorithm uses.
2. **Time Complexities as a Big-O Notation:**

   Learn implications of below listed Notations:
   - **Constant Time (O(1))**
   - **Linear Time (O(n))**
   - **Logarithmic Time (O(log n))**
   - **Linearithmic Time (O(n log n))**
   - **Quadratic Time (O(n^2))**

Time Complexity

O(1)
O(n)
O(log n)
O(n log n)
O(n^2)

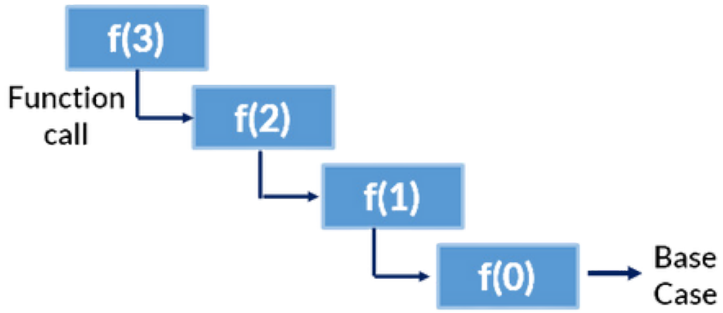


Space Complexity

5 Iteration and Recursion

Iteration is a process where a a block of code is repeatedly executed until a certain condition is met whereas, **Recursion** is a process where a function calls itself directly or indirectly in order to solve a problem.

1. **Iteration:**
   - Learn how iteration affects control flow of a program.
   - Recognize scenarios where iteration or recursion might be more suitable.
   - Explore Iterative vs. Recursive Approaches.
2. **Recursion:**
   - Understand **recursion** as a problem-solving technique.
   - Recognize the importance of **recursive functions.**
   - Understand **tail recursion** and its optimization.



Questions to Master Recursion and Backtracking

Question 2: Combination Sum

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order. The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different. The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Question 3: Combination Sum II

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.
Each number in candidates may only be used once in the combination.

Questions to Master Recursion and Backtracking

Question 4: Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

6 Arrays

An **array** is a collection of elements of the same data type, stored at contiguous memory locations. Imagine it like a row of boxes, each holding data of the same kind.

Here's a roadmap to master arrays:

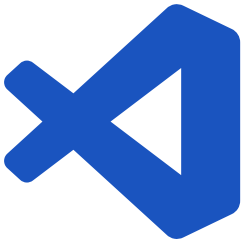
1. **Introduction to Arrays**
    - Learn how to declare and initialize arrays in Java.
    - Learn how elements in an array are accessed.
    - Understand how arrays are stored in memory.
2. **Operations on Arrays**

- Perform different operations on Arrays such as **Traversal, Insertion, Deletion and Searching**.

3. **Dynamic Arrays**

- Understand the limitations of static arrays.
- Learn about dynamic arrays like ArrayList in Java





Questions to Master Arrays

Question 1: Rotate Arrays

Given an integer array **nums**, rotate the array to the right by **k** steps, where **k** is non-negative.

Question 2: Squares of a Sorted Array

Given an integer array **nums** sorted in *non-decreasing* order, return an array of the *squares of each number* sorted in non-decreasing order.

Question 3: Maximum Product Subarray

Given an integer array nums, find a
subarray that has the largest product, and return the product.

Question 4: Kadane's Algo

Given an integer array nums, find a
subarray that has the largest product, and return the product.

Questions to Master Arrays

Question 5: Majority Element

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Question 6: Next Greater Element III

Given a positive integer n, find the smallest integer which has exactly the same digits existing in the integer n and is greater in value than n. If no such positive integer exists, return -1.

Question 7: Max Chunks to make Sorted

You are given an integer array arr of length n that represents a permutation of the integers in the range [0, n - 1]. We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return the largest number of chunks we can make to sort the array.

Questions to Master Arrays

Question 8: Number of Subarrays with Bounded Maximum

Given an integer array nums and two integers left and right, return the number of contiguous non-empty subarrays such that the value of the maximum array element in that subarray is in the range [left, right].

Question 9: First Missing Positive

Given an unsorted integer array nums, return the smallest missing positive integer. You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.

Question 10: Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

7 Strings

A String represents a sequence of characters, typically stored as an array of characters in memory.

Here's a roadmap to master strings:

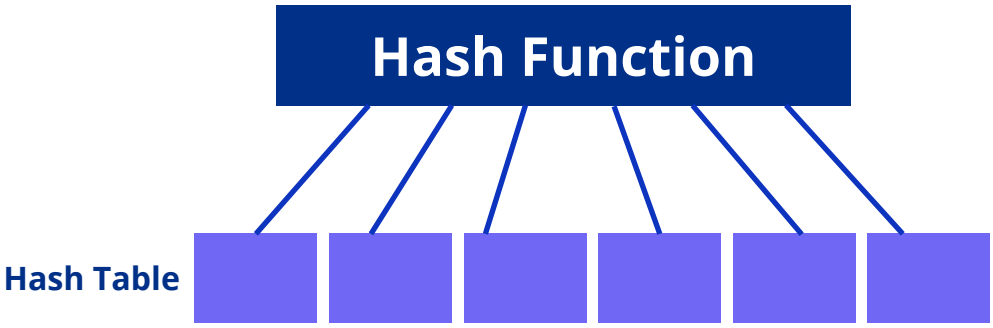
1. **Introduction to Strings**
   - Learn the difference between immutable and mutable string types.
   - Learn how to declare and initialize strings.
2. **String Manipulation**
   - **Concatenation:** Explore various methods for concatenating strings.
   - **Substring:** Understand the concept of substrings within a string.
   - **Palindrome Check:** Implement algorithms to check if a given string is a palindrome.

8 Hashing and Hash Functions

Hashing involves mapping data of arbitrary size to fixed-size values, typically for the purpose of quick data retrieval. It is commonly used to implement data structures like hash tables.

Here's a roadmap to master Hashing:

- Understand what **hashing** is and how it involves transforming data into a fixed-size value.
- Learn about **hash functions** that map data to hash codes.
- Learn about **collision resolution** techniques, such as chaining and open addressing.
- Understand how **chaining** handles collisions.
- Learn about **open addressing** techniques, including linear probing and quadratic probing.



Questions to Master HashMap

Question 1: Subarray Sum Equals K

Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k.

A subarray is a contiguous non-empty sequence of elements within an array.

Question 2: Subarray Sums Divisible by K

Given an integer array nums and an integer k, return the number of non-empty subarrays that have a sum divisible by k. A subarray is a contiguous part of an array.

Question 3: Insert Delete GetRandom O(1)

There is a 2D grid of size n x n where each cell of this grid has a lamp that is initially turned off.

Question 4: Longest Consecutive Sequence

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

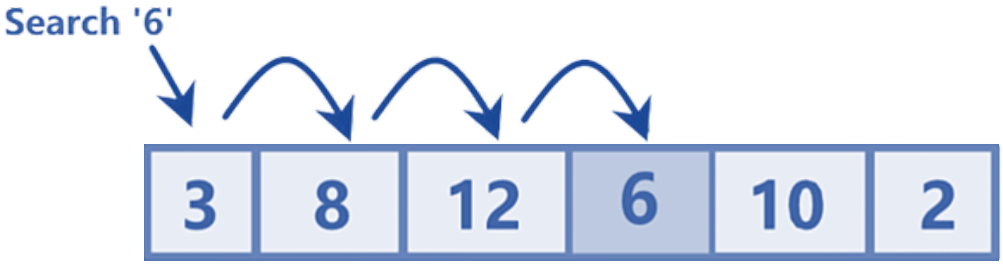
You must write an algorithm that runs in O(n) time.

9 Searching Algorithms

Searching algorithms are techniques used to locate a specific element within a data structure.

Here's a roadmap for searching algorithms in DSA:

1. **Linear Search**: Works by iterating through each element in the data structure until the target element is found.
2. **Binary Search:** Repeatedly divides the search space in half based on the comparison with the middle element.
3. **Interpolation Search:** Variation of binary search that estimates the position of the target element based on its value and the distribution of data.
4. **Exponential Search:** Used particularly when the data is unbounded or has an unknown size



Questions to Master Binary Search

Question 1: Capacity to ship Packages within D Days

A conveyor belt has packages that must be shipped from one port to another within days days.

The ith package on the conveyor belt has a weight of weights[i]. Each day, we load the ship with packages on the conveyor belt (in the order given by weights). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within days days.

Question 2: Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Questions to Master Binary Search

Question 3: Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with distinct values). Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

Question 4: Search in Rotated Sorted Array II

There is an integer array nums sorted in non-decreasing order (not necessarily with distinct values).

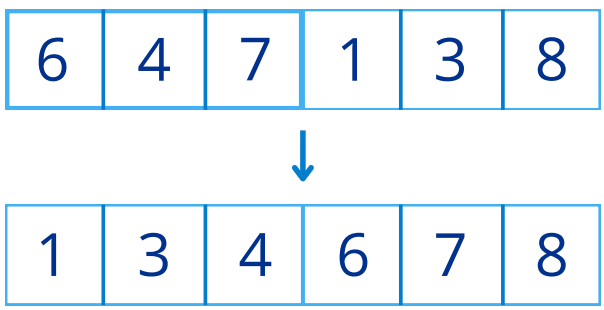
Before being passed to your function, nums is rotated at an unknown pivot index k (0 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,4,4,5,6,6,7] might be rotated at pivot index 5 and become [4,5,6,6,7,0,1,2,4,4]. Given the array nums after the rotation and an integer target, return true if target is in nums, or false if it is not in nums.

10 Sorting Algorithms

Sorting is used to rearrange elements according to a specific order, usually ascending or descending.

Here's a roadmap for sorting algorithms in DSA:

1. **Bubble Sort**: compares and swaps adjacent elements until the entire list is sorted.
2. **Selection Sort:** works by selecting the smallest (or largest) element in each iteration.
3. **Insertion Sort:** Builds sorted list one element at a time.
4. **Merge Sort:** divides the list into halves, sorts each half, and merges them back together.
5. **Quick Sort:** partitions the array and recursively sorts its subarrays.
6. **Heap Sort:** uses a binary heap data structure to sort elements.



11 Linked Lists

Linked List store elements in nodes, each containing data and a pointer to the next node in the list.

Here's a roadmap for Linked List in DSA:

1. **Types of Linked List**
   - **Singly Linked List:** Each node has only one pointer to the next node.
   - **Doubly Linked List:** Each node has two pointers, one to the next node and one to the previous node.
   - **Circular Linked List:** The last node points back to the first node, creating a loop.
2. **Linked List Operations**
   - **Traversal:** Visiting each node in the list sequentially.
   - **Insertion:** Adding a new node at a specific position (beginning, end, or middle).
   - **Deletion:** Removing a node from a specific position.

Questions to Master Linked List

Question 1: Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Question 2: Longest Consecutive Sequence

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the LRUCache class:

- LRUCache(int capacity) Initialize the LRU cache with **positive** size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, **evict** the least recently used key.

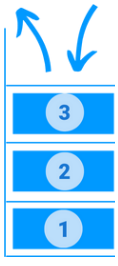
The functions get and put must each run in O(1) average time complexity.

12 Stacks

Stack follows **Last In, First Out (LIFO)** principle. This "last in, first out" behavior dictates how elements are added and removed from the stack.

Here's a roadmap for Stack in DSA:

1. **Introduction to Stack**
   - Understand what a stack is and its Last In, First Out (LIFO) principle.
   - Learn how stacks are used to evaluate arithmetic expressions.
2. **Stack Operations**
   - **Push:** adds a new element to top of the stack.
   - **Pop**: removes and returns the element currently at the top of the stack.
   - **Peek**: view the element at the top of the stack without removing it.

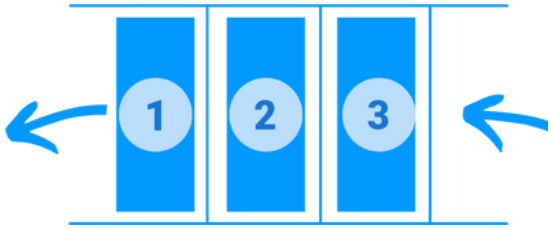


13 Queue and Dequeue

Queue follows the **First In, First Out (FIFO)** principle, similar to a waiting line. The element that enters the queue first is the first one to be removed.

Here's a roadmap for Stack in DSA:

1. **Introduction to Queue**
   - Understand what a queue is and FIFO principle.
   - Familiarize with types of Queue (Circular and Priority Queue).
2. **Queue Operations**
   - **Enqueue:** Adds a new element to back of queue.
   - **Dequeue:** Removes and returns element from the front of the queue.
   - **Front and Rear:**
3. **Introduction to Dequeue**
   - Learn about double-ended queues that support insertion and deletion at both ends.



Questions to Master Stack and Queue

Question 1: Next Greater Element I

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.

Question 2: Largest Rectangle in Histogram

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

Question 3: Maximal Rectangle

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Questions to Master Stack and Queue

Question 4: Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Question 5: Basic Calculator

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Question 6: Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

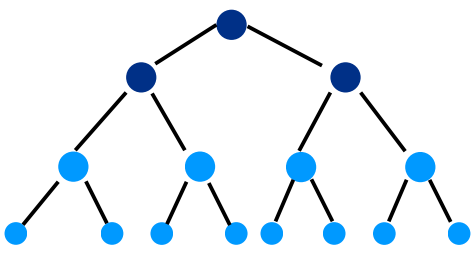
- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

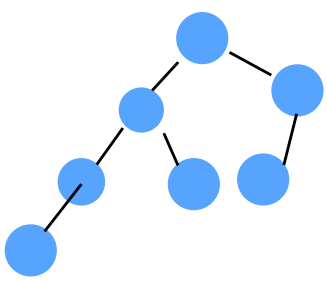
14 Trees

Trees are non-linear data structures, represents hierarchical relationships between elements.

Here's a roadmap for Trees in DSA:

1. **Introduction to Trees**
   - Learn about nodes, edges, root, leaves, parent, child, siblings, etc.
2. **Binary Trees**
   - Understand binary trees where each node has at most two children.
   - These are the **types**: Full Binary Tree, Complete Binary Tree, Perfect Binary Tree.
   - Implement Tree **Traversal** (Inorder, Preorder, Postorder).
3. **Binary Search Tree**
   - Understand the concept of binary search trees and Implement different operations on BST.





Question to Master Trees

Question 1: Binary Tree Preorder Traversal
Given the root of a binary tree, return the preorder traversal of its nodes' values.

Question 2: Binary Tree Inorder Traversal
Given the root of a binary tree, return the inorder traversal of its nodes' values.

Question 3: Binary Tree Postorder Traversal
Given the root of a binary tree, return the postorder traversal of its nodes' values.

Question 4: Binary Tree Right Side View
Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

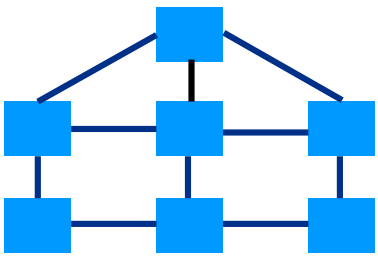
Question 5: Construct Binary Tree from Preorder and Inorder Traversal
Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

15 Heap

Heaps are tree-based data structures known for their retrieval of maximum or minimum elements. They follow a specific heap property, ensuring a specific order within tree.

Here's a roadmap for Heap in DSA:

1. **Introduction to Heap**
   - Understand what a heap is and its basic properties.
   - Types of Heap: Min Heap and Max Heap.
2. **Heap Operations**
   - **Insertion :** Insert elements into a heap while maintaining the heap property.
   - **Extraction:** Understand the process of extracting the minimum (or maximum) element from a heap.
3. **Applications of Heap**
   - **Priority Queue:** Understand how heaps are used to implement efficient priority queues.
   - **Heap Sort:** Learn how heapsort utilizes the heap data structure for sorting elements.



Question to Master Heap

Question 1: Kth Largest Element

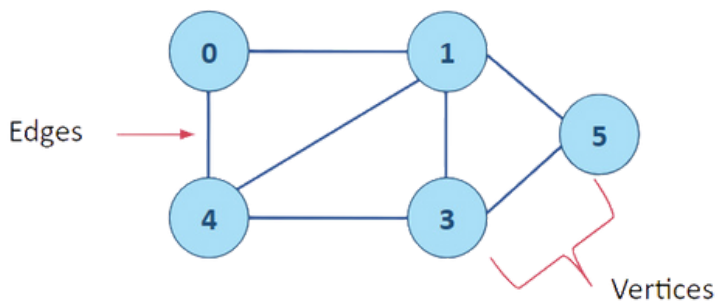
Given an integer array nums and an integer k, return the kth largest element in the array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

16 Graphs (Basics)

Graphs are used to represent relationships between pairs of objects. Graphs are composed of vertices (nodes) and edges (connections between nodes).

Here's a roadmap for Graphs in DSA:

1. **Introduction to Graphs**
   - Understand graphs including vertices, edges, directed vs. undirected graphs, weighted vs. unweighted graphs, etc.
2. **Graph Representation**
   - **Adjacency Matrix:** two-dimensional array (matrix) where each cell indicates whether there is an edge between two vertices.
   - **Adjacency List:** store the graph as an array of lists where each list represents the neighbors of a vertex



Question to Master Graphs

Question 1: Rotting Oranges

You are given an m x n grid where each cell can have one of three values:

- 0 representing an empty cell,
- 1 representing a fresh orange, or
- 2 representing a rotten orange.

Every minute, any fresh orange that is 4-directionally adjacent to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return -1

Question 2: Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Question to Master Graphs

Question 3: O 1 Matrix

Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Question 2: Redundant Connection

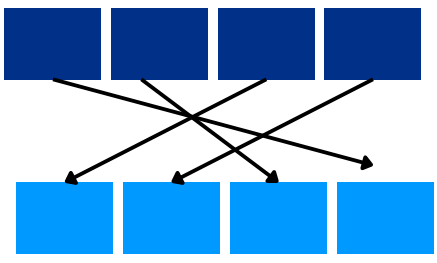
In this problem, a tree is an undirected graph that is connected and has no cycles.

You are given a graph that started as a tree with n nodes labeled from 1 to n, with one additional edge added. The added edge has two different vertices chosen from 1 to n, and was not an edge that already existed. The graph is represented as an array edges of length n where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the graph.

Return an edge that can be removed so that the resulting graph is a tree of n nodes. If there are multiple answers, return the answer that occurs last in the input.

17 Sorting and Searching Algorithms (Advanced)

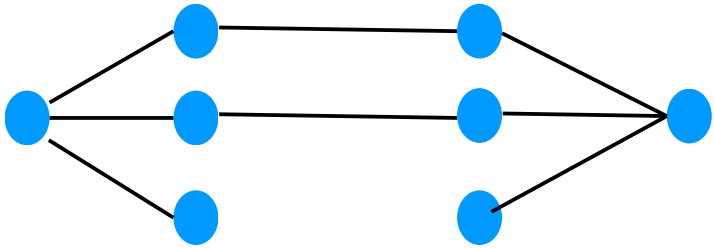
1. **Advanced Searching Algorithms**
   - **Jump Search:** learn how it works by jumping ahead by fixed steps and then performing linear search in the smaller interval.
   - **Fibonacci Search:** learn how it works by dividing the array into Fibonacci subarrays and recursively searching within these subarrays.
2. **Advanced Sorting Algorithms**
   - **Counting Sort:** learn how Counting Sort works by counting the occurrences of each unique element and then placing them in the correct order.
   - **Radix Sort:** learn how Radix Sort works by repeatedly sorting elements based on each digit or character position, from the least significant to the most significant.



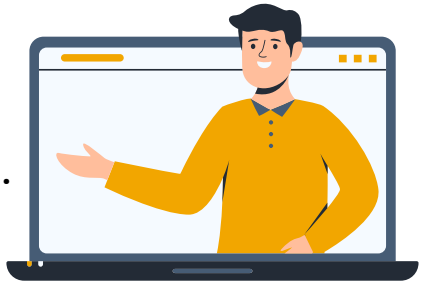
18 Necessary Algorithms

Following are three fundamental techniques in the field of algorithms and are included in a DSA roadmap.

1. **Dynamic Programming**
   - Learn about the principle of optimality and how to design dynamic programming algorithms using either top-down (memoization) or bottom-up (tabulation) approaches.
2. **Greedy Algorithms**
   - Learn about the greedy-choice property and how to design greedy algorithms by selecting the best choice at each step without reconsideration.
3. **Backtracking**
   - Learn about the recursive nature of backtracking algorithms and how to design them to efficiently explore the solution space.



Data Structures Tutorial



ScholarHat offers concise, insightful DSA articles. Dive into DSA with clear explanations and practical examples, perfect for enhancing your programming skills.

- Learn Sorting Algorithms
- Learn Trees:
  - Trees in DSA
  - Segment Trees
  - AVL Trees
  - Spanning Trees
- Learn Hash Table
- Learn Linked List
- Learn Stack
- Learn Queue
- Learn Heap
- Learn Graphs
- Learn Greedy Algorithm

How to follow this roadmap?

At ScholarHat, we believe **mastering a technology** is a **three-step process** as mentioned below:

- **Step1 - Learn Skills:** You can learn Data structures and Algorithm **through Videos** on YouTube or **Videos based courses**. For topic revision and recalling make **short notes**. Solve Algorithmic Challenges on Platforms like **LeetCode**, **HackerRank**, etc.
- **Step2 - Build Experience:** You can build hands-on experience by creating **workflow** using Data Structures and Algorithms like **Mini Calculator**, **Fitness Application**, **Banking System** etc.
- **Step3 - Empower Yourself:** Build your **strong profile** by mentioning all the above skills with **hands-on experience** on Data Structures. Prepare yourself with interview **Q&A about DSA** to crack your next job interview.

Congrats!

You are just one interview away!

WAS THIS HELPFUL?

Share with your friend who needs it!

Love. Like. Comment. Share.

Learn. Build. Empower.