LMNs- Algorithms

Last Updated : 22 Jan, 2025

An **algorithm** is a step-by-step procedure or set of rules to solve a specific problem. It is a logical sequence of instructions designed to achieve a desired output for given inputs.

Analysis of Algorithm

1) **Worst Case Analysis (Usually Done)**: In the worst case analysis, we calculate upper bound on running time of an algorithm by considering worst case (a situation where algorithm takes maximum time)

2**) Average Case Analysis (Sometimes done)**: In average case analysis, we take all possible inputs and calculate computing time for all of the inputs.

3)**Best Case Analysis (Bogus)**: In the best case analysis, we calculate lower bound on running time of an algorithm.

Table of Content

Asymptotic Notations

- **Θ Notation:** The theta notation bounds a functions from above and below, so it defines exact asymptotic behavior.

Θ((g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that
*0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}*

- **Big O Notation:** The Big O notation defines an upper bound of an algorithm, it bounds a function only from above.

O(g(n)) = { f(n): there exist positive constants c and n0 such that
0 <= f(n) <= cg(n) for all n >= n0}

- **Ω Notation:** Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.

Ω (g(n)) = {f(n): there exist positive constants c and n0 such that
0 <= cg(n) <= f(n) for all n >= n0}.

read more about - Asymptotic Notations

Solving recurrences

- **Substitution Method**: We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.
- **Recurrence Tree Method:** We draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work done at all levels.
- **Master theorem Method:** Only for following type of recurrences or for recurrences that can be transformed to following type.

T(n) = aT(n/b) + f(n) where a >= 1 and b > 1

Sorting

Algorithm	Worst Case	Average Case	Best Case	Min. no. of swaps	Max. no. of swaps
Bubble	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	0	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	0	$\Theta(n)$
Insertion	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	0	$\Theta(n^2)$

Quick	$\Theta(n^2)$	$\Theta(nlgn)$	$\Theta(nlgn)$	0	$\Theta(n^2)$
Merge	$\Theta(nlgn)$	$\Theta(nlgn)$	$\Theta(nlgn)$	Is not in-place sorting	Is not in-place sorting
Heap	$\Theta(nlgn)$	$\Theta(nlgn)$	$\Theta(nlgn)$	$O(nlgn)$	$\Theta(nlgn)$

Searching

Algorithm	Worst Case	Average Case	Best Case
Linear Search	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
Binary Search	O (logn)	O (logn)	O (1)

Trees

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures. Depth First

Important Tree Properties and Formulas

(a) Inorder
(b) Preorder
(c) Postorder

Binary Search Tree

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

1. Insertion
2. Deletion

AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

1. Insertion
2. Deletion

B-Tree

B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like **B-Tree** and Red Black Trees), it is assumed that everything is in main memory. To understand use of B-Trees, we must think of huge amount of data that cannot fit in main memory. When the number of keys is high, the data is read from disk in the form of blocks. Disk access time is very high compared to main memory access time. The main idea of using B-Trees is to reduce the number of disk accesses.

Properties of Prim's Minimum Spanning Tree Algorithm,

1. B-Tree Insertion
2. B-Tree Deletion

Minimum Spanning Tree

Minimum Spanning Tree (MST) problem: Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices. MST is fundamental problem with diverse applications.

- Network design– telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems – traveling salesperson problem, Steiner tree
- Cluster analysis- k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.

Example: Kruskal's Minimum Spanning Tree Algorithm

Graph

Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.

2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost. Following two are the most commonly used representations of graph.

1. **Adjacency Matrix**: Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
2. **Adjacency List :** An array of linked lists is used. Size of the array is equal to number of vertices.

Graph Algorithms

Algorithm	**Time Complexity**
Breadth First Traversal for a Graph	O(V+E) for adjacency list representation and O(V * V) for adjacency matrix representation.
Depth First Traversal for a Graph	O(V+E) for adjacency list representation and O(V * V) for adjacency matrix representation.
Dijkstra's shortest path algorithm	Adjacency matrix- O(V^2). Adjacency list- O(E log V)
Topological Sorting: Shortest Path in Directed Acyclic Graph	O(V+E)

Some Interesting Graph Questions

Divide and Conquer

1. ***Divide:*** Break the given problem into subproblems of same type.
2. ***Conquer****:* Recursively solve these subproblems
3. ***Combine:*** Appropriately combine the answers

Time Complexity Analysis

If a problem of size `n` is divided into `k` subproblems, each of size `n/m`, and the merging step takes `O(f(n))`, the time complexity is given by the **Master Theorem**:

T(n) = kT(n/m) + O(f(n))

Following are some standard algorithms that are Divide and Conquer algorithms.

1. Binary Search

Binary Search is a searching algorithm. This algorithm include following steps:

- Divide the search space into two halves.
- Compare the middle element with the target.
- Recursively search in the relevant half.

Time Complexity: O(log n)

read more

2. Quicksort

It is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the subarrays on left and right of pivot element.

Time Complexity: Best and Average Case: O(n log n), Worst Case: $O(n^2)$ (when the pivot is poorly chosen)

read more

3. Merge Sort

It is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.

Time Complexity: Worst, Average, and Best Case: O(n log n)

read more

4. Closest Pair of Points

The problem is to find the closest pair of points in a set of points in x-y plane. The problem can be solved in O(n^2) time by calculating distances of every pair of points

and comparing the distances to find the minimum. The Divide and Conquer algorithm solves the problem in O(nLogn) time.

5. Strassen's Matrix Multiplication

- **Steps**:
  1. Divide matrices into smaller submatrices.
  2. Perform recursive multiplications and additions.
  3. Combine results.

- **Time Complexity**: $O(n^{2.81})$

Greedy Approach

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems. An optimization problem can be solved using Greedy if the problem has the following property:

At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem.

Following are some standard algorithms that are Greedy algorithms:

1) Kruskal's Minimum Spanning Tree (MST)

In Kruskal's algorithm, we create a MST by picking edges one by one. The Greedy Choice is to pick the smallest weight edge that doesn't cause a cycle in the MST constructed so far.

Time Complexity: O(E log V).

2) Prim's Minimum Spanning Tree

In Prim's algorithm also, we create a MST by picking edges one by one. We maintain two sets: set of the vertices already included in MST and the set of the vertices not yet included. The Greedy Choice is to pick the smallest weight edge that connects the two sets.

Time Complexity: O(E log V).

3) Dijkstra's Shortest Path

The Dijkstra's algorithm is very similar to Prim's algorithm. The shortest path tree is built up, edge by edge. We maintain two sets: set of the vertices already included in the tree and the set of the vertices not yet included. The Greedy Choice is to pick the edge that connects the two sets and is on the smallest weight path from source to the set that contains not yet included vertices.

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

4) Huffman Coding

Huffman Coding is a loss-less compression technique. It assigns variable length bit codes to different characters. The Greedy Choice is to assign least bit length code to the most frequent character.

Time Complexity:O(n log n).

read more about - Greedy Approach

Dynamic Programming

Dynamic Programming (DP) is a problem-solving approach that breaks a problem into smaller overlapping subproblems, solves each subproblem once, and stores its solution to avoid redundant computations. This technique is especially useful for optimization problems

Steps in Dynamic Programming

- **Define Subproblems**: Break the problem into smaller, overlapping subproblems.
- **Recurrence Relation**: Define the relation that connects the solution of the problem with its subproblems.
- **Base Case**: Identify the simplest subproblems and their solutions.
- **Solve and Store**: Use memoization or tabulation to store solutions to subproblems.
- **Construct Solution**: Combine solutions of subproblems to solve the overall problem.

Common Examples of Dynamic Programming

1. Fibonacci Numbers

- **Problem**: Compute the nth Fibonacci number.
- **Recurrence Relation**: `F(n) = F(n-1) + F(n-2)`
- **Time Complexity**: O(n) with memoization or tabulation.

2. Longest Common Subsequence (LCS)

- **Problem**: Find the longest subsequence common to two strings.
- **Recurrence Relation**: `LCS(X, Y, i, j) = 1 + LCS(X, Y, i-1, j-1)` if `X[i] == Y[j]`. Otherwise, `LCS(X, Y, i, j) = max(LCS(X, Y, i-1, j), LCS(X, Y, i, j-1))`
- **Time Complexity**: O(m * n) for strings of lengths `m` and `n`.

3. Knapsack Problem

- **Problem**: Maximize the total value of items that can be put in a knapsack of a given capacity.
- **Recurrence Relation**: `K(i, W) = max(K(i-1, W), K(i-1, W-w[i]) + v[i])`. If item `i` is included, add its value `v[i]` and reduce capacity `W` by its weight `w[i]`. Otherwise, skip the item.
- **Time Complexity**: O(n * W), where `n` is the number of items and `W` is the knapsack capacity.

4. Matrix Chain Multiplication

- **Problem**: Find the most efficient way to multiply matrices by minimizing the number of scalar multiplications.
- **Recurrence Relation**: `M(i, j) = min(M(i, k) + M(k+1, j) + p[i-1]*p[k]*p[j])` for all `k` from `i` to `j-1`
- **Time Complexity**: $O(n^3)$.

read more about - Dynamic Programming

Backtracking

Backtracking is a general algorithmic technique that involves exploring all possible solutions to a problem by building a solution incrementally and abandoning solutions that fail to satisfy the constraints. It is often used for solving problems that require finding combinations, permutations, or subsets.

Steps in Backtracking

1. **Choose**: Make a choice for the current step.
2. **Explore**: Recursively explore all possible solutions based on the current choice.

3. **Backtrack**: If the solution is invalid or all possibilities have been explored, undo the last choice and try a different one.

Examples of Backtracking

1. N-Queens Problem

- **Problem**: Place `N` queens on an `N x N` chessboard so that no two queens threaten each other.
- **Approach**: Place one queen in each row and recursively check if the placement is valid.

2. Sudoku Solver

- **Problem**: Fill a 9x9 grid with numbers 1-9 such that each row, column, and 3x3 sub-grid contains all numbers without repetition.
- **Approach**: Place a number in an empty cell and recursively check if the grid is valid.

3. Subset Sum Problem

- **Problem**: Determine if there is a subset of a given set whose sum equals a target value.
- **Approach**: Either include or exclude an element in the subset and recursively check for the solution.

4. Generating Permutations

- **Problem**: Generate all possible permutations of a given string or array.
- **Approach**: Swap elements recursively and backtrack to undo the swaps.

read more about - Backtracking

Similar Reads

LMNs-Data Structures

Data structures are ways to organize and store data so it can be used efficiently. They are essential in computer science for managing and processing information in...

14 min read

Top 20 Greedy Algorithms Interview Questions

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy...

1 min read

Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of...

15+ min read

Data Structures and Algorithms | Set 36

Que - 1. The function shiftNode() which takes as input two linked lists- destination and source. It deletes front node from source and places it onto the front of destination....

4 min read

Data Structures and Algorithms | Set 37

Que - 1. For 8 keys and 6 slots in a hashing table with uniform hashing and chaining, what is the expected number of items that hash to a particular location. (A) 2.33 (B)...

4 min read

Classification of Routing Algorithms

Pre-requisites: Difference between Static and Dynamic RoutingRouting is the process of establishing the routes that data packets must follow to reach the destination. In th...

8 min read

Mutation Algorithms for Real-Valued Parameters (GA)

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. In each generation chromosomes(our solution...

3 min read

Interesting Examples of algorithms in everyday life

Ever found shortest path from Place A to Place B on Google Maps? Ever rolled a dice just by a click in an online game? Ever used search functionality in a website? One thi...

2 min read

Mutation Algorithms for String Manipulation (GA)

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. In each generation chromosomes(our solution...

2 min read

Algorithms | Recurrences | Set 1

Question 1: Which of the following is the value of T3(n) where T3(n) is defined as T3(n) = 5*T3(n-1) - 4*T3(n-2) C1*5n + C2*4n C1 + C2*4n C1*2n + C2*4n C1*5n +...

5 min read

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Python Tutorial

Python Programming Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Commerce

Accountancy
Business Studies
Economics
Management
HR Management
Finance
Income Tax

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Resume Templates
Aptitude Preparation
Puzzles
Company-Wise Preparation
Companies
Colleges

Competitive Exams

JEE Advanced
UGC NET
UPSC
SSC CGL
SBI PO
SBI Clerk
IBPS PO
IBPS Clerk

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets
Recent Articles

Free Online Tools

Typing Test
Image Editor
Code Formatters

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write

Code Converters
Currency Converter
Random Number Generator
Random Password Generator

Share your Experiences
Internships

DSA/Placements

DSA - Self Paced Course
DSA in JavaScript - Self Paced Course
DSA in Python - Self Paced
C Programming Course Online - Learn C with Data Structures
Complete Interview Preparation
Master Competitive Programming
Core CS Subject for Interview Preparation
Mastering System Design: LLD to HLD
Tech Interview 101 - From DSA to System Design [LIVE]
DSA to Development [HYBRID]
Placement Preparation Crash Course [LIVE]

Development/Testing

JavaScript Full Course
React JS Course
React Native Course
Django Web Development Course
Complete Bootstrap Course
Full Stack Development - [LIVE]
JAVA Backend Development - [LIVE]
Complete Software Testing Course [LIVE]
Android Mastery with Kotlin [LIVE]

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Mastering Generative AI and ChatGPT
Data Science Course with IBM Certification

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

GATE

GATE CS & IT Test Series - 2025
GATE DA Test Series 2025
GATE CS & IT Course - 2025
GATE DA Course 2025
GATE Rank Predictor