How to learn DSA?

1. Learn at least one programming language (C++ , Java , Python , or JavaScript) and develop your basic logic.
2. Discover the complexities of time and space
3. Learn data structures and algorithms
4. DSA Practice
5. Solve problems every day using GfG POTD, every week using GfG Weekly Contest , and every month using GfG Job-A-Thon .

Hopefully you've mastered the programming language of your choice, here's the next step on the journey: discover the complexities of time and space.

1. Construction of logic

Once you have learned the basics of a programming language, it is recommended to learn the construction of basic logic

- Logical construction problems
- Practical problems on mathematical algorithms
- Logic Construction Quiz

2. Discover the complexities

To analyze algorithms, we mainly measure the order of growth of time or space taken in terms of input size. We do this in the worst case scenario in most cases. Please refer to the links below for a clear understanding of these concepts.

- Complexity Analysis Guide
- Complexity Analysis Quiz

3. Matrix

An array is a linear data structure in which elements are assigned **contiguous memory** , allowing **constant-time access** .

- Array Data Structure Guide
- Practical Array Problems
- Top 50 Array Coding Problems for Interviews
- Array Quiz

4. Search Algorithms

Search algorithms are used to locate specific data within a large data set. They help **to find a target value** within the data. There are various types of search algorithms, each with its own approach and efficiency.

- Search Algorithms Guide
- Practical Research Problems
- Top 50 Coding Problems to Look for in Interviews
- Research Quiz

5. Sorting Algorithm

Sorting algorithms are used to **arrange** the elements of a list in a **specific order** , such as numeric or alphabetical. It organizes the elements in a systematic way, making it easier to find and access specific elements.

- Guide to Sorting Algorithms
- Practical problems on sorting algorithm
- Most Common Sorting Interview Questions and Problems
- Sorting Quiz

6. Hashing

Hashing is a technique that generates a fixed-size output (hash value) from a variable-size input using mathematical formulas called hash functions. Hashing is commonly used in data structures for efficient lookups, insertions, and deletions.

- Guida all'hashing
- Practical Hashing Problems
- Top 20 Hashing Interview Questions
- Hashing Quiz

7. Two-pointer technique

In the two-pointer technique, we usually use two index variables from two corners of an array. We use the two-pointer technique to search for a required point or value in an array.

- Two pointer technique
- Two Pointer Practice Problems

- Quiz on the two aiming techniques

8. Window Sliding Technique

In Window Sliding technique, we use the result of the previous subarray to quickly calculate the result of the current one.

- Window Sliding Technique
- Practical problems on the sliding window
- Sliding Window Quiz

9. Prefix Sum Technique

In the prefix sum technique, we calculate the sums of the prefixes of an array to quickly find the results for a subarray.

- Prefix Sum Technique
- Practical problems on the sum of prefixes
- Prefix Sum Quiz

10. Rope

A string is a sequence of characters, usually immutable and with a limited set of elements (lowercase letters or all English alphabets).

- Guide on the ropes
- Practical problems on the rope
- Top 50 String Encoding Problems for Interviews
- String Quiz

11. Recursion

Recursion is a programming technique in which a function **calls itself** within its own definition. It is usually used to solve problems that can be broken down into smaller instances of the same problem.

- Guide to Recursive Algorithms
- Practical problems on recursion algorithm
- Top 50 Recursion Algorithm Problems for Interview
- Recursion Quiz

12. Matrix/Grid

Matrix is a two-dimensional array of elements, arranged in **rows** and **columns** . It is represented as a rectangular grid, with each element at the intersection of a row and a column.

- Matrix Data Structure Guide
- Practical problems on matrix/grid
- Top 50 Matrix/Grid Interview Problems
- Matrix/Grid Quiz.

13. How many

Stack is a linear data structure that follows the **Last In, First Out (LIFO)** principle . Stacks play an important role in managing function calls, memory, and are widely used in algorithms such as the stock span problem, next greatest element, and largest area in a histogram.

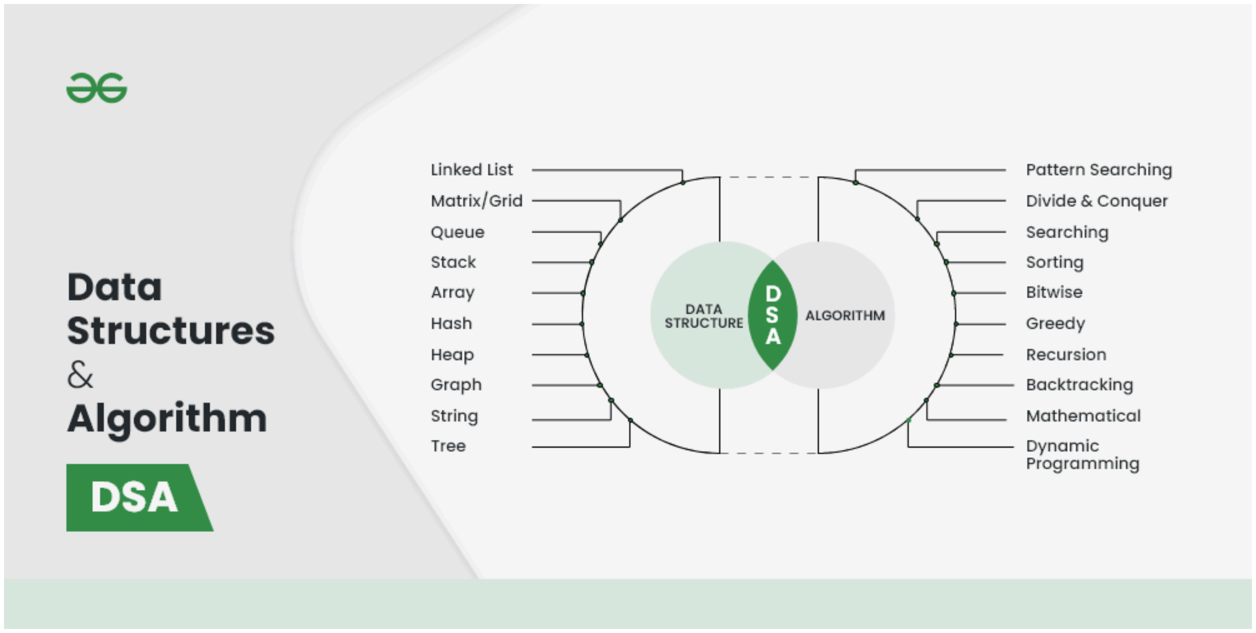
- Stack Data Structure Guide

- Stack Practice Problems
- Top 50 Stack Interview Problems
- Quiz su Stack

14. Coda

A **queue** is a linear data structure that follows the **First In, First Out (FIFO)** principle . Queues play an important role in managing tasks or data in order, scheduling, and message handling systems.

- Queue Data Structure Guide
- Practical problems in queue
- Top 50 issues in the interview queue
- Quiz in queue



15. Linked List

Linked list is a linear data structure that stores data in nodes, which are connected by pointers. Unlike arrays, nodes of linked lists are not stored in contiguous memory locations and can **only be accessed sequentially** , starting from the head of the list.

- Linked List Data Structure Guide
- Practical problems on linked lists
- Top 50 Problems in the Linked List for Interviews
- Linked List Quiz

16. Tree

Tree is a **non lineare, gerarchico** data structure consisting of nodes connected by edges, with a top node called **root** e nodi con nodi figlio. È ampiamente utilizzato nei **file system** , **database** , **algoritmi decisionali** , ecc.

- Tree Data Structure Guide
- Practical tree problems
- Top 50 Tree Coding Problems for Interviews
- Tree Quiz

17. Heap

Heap is a albero binario completo data structure che soddisfa la **proprietà heap** . Gli heap sono comunemente usati per implementare **le code di priorità** , dove l' elemento **più piccolo** o **più grande** è sempre alla radice dell'albero.

- Heap Data Structure Guide

- Practical problems on Heap
- Top 50 Heap Problems for Interviews
- Quiz su Heap

18. Graph

A graph is a **non lineare** data structure consisting of a finite set of **vertici** (o nodi) e un insieme di **spigoli** (o collegamenti) che collegano una coppia di nodi. I grafici sono ampiamente utilizzati per rappresentare relazioni tra entità.

- Guide on graphic algorithms
- Practical problems on the graph
- Top 50 Interview Chart Problems
- Graph Quiz

19. Greedy algorithm

The Greedy algorithm builds the solution piece by piece and chooses the next piece that provides the most obvious and immediate benefit, i.e., the **most optimal choice at that time** . So problems where the **locally optimal** choice also leads to global solutions are best suited for Greedy.

- Guide to Greedy Algorithms
- Practical problems on the Greedy algorithm
- Top 20 Greedy Algorithms Interview Questions
- Maybe your Greedy

20. Dynamic programming

Dynamic programming is a method used to solve complex problems by breaking them into più semplice**subproblems . By solving each subproblem una sola volta** e **memorizzando i risultati** , evita calcoli ridondanti, portando a **soluzioni più efficienti** per un'ampia gamma di problemi.

- Dynamic Programming Guide
- Practical problems on dynamic programming
- Top 50 Dynamic Programming Interview Problems
- Quiz your DP

21. Other algorithms

Bitwise algorithms: operate on single bits of numbers.

- Bitwise Algorithms Guide
- Practical problems on Bit Magic
- Bit Magic Quiz

Backtracking algorithm: Follows recursion with the option to **go back and retrace the track** if the solution from the current point is not feasible.

- Tornando indietroAlgorithms Guida
- Practical problems on backtracking algorithm
- Top 20 Backtracking Algorithm Interview Questions
- Quiz sul Backtracking

Divide and conquer: A strategy for solving problems by dividing them into **smaller subproblems** , solving those subproblems, and combining the solutions to obtain the final solution.

- Guide to the Divide and Conquer algorithm
- Practical problems on the Divide and Conquer algorithm
- Quiz on Divide and Conquer

Branch and Bound: Used in combinatorial optimization problems to systematically search for the best solution. It works by dividing the problem into smaller subproblems, or branches, and then eliminating some branches based on the bounds of the optimal solution. This process continues until the best solution is found or all branches have been explored.

- Guida all'algoritmo Branch and Bound

Geometric algorithms are a set of algorithms that solve problems involving **shapes** , **points** , **lines** , and polygons.

- Guide to Geometric Algorithms
- Practical problem on geometric algorithms

Randomized algorithms are algorithms that use **randomness** to solve problems. They use random inputs to achieve their goals, often resulting in **più semplice** ed efficiente**solutions**Questi algoritmi potrebbero **non produrre lo stesso risultato,** ma sono particolarmente utili in situazioni in cui un **approccio probabilistico** è accettabile.

- Guide to Randomized Algorithms

Memorandum

- Blind 75
- SDE SHEET – A Complete Guide to SDE Preparation
- GeeksforGeeks Master Sheet – List of all Cheat Sheets