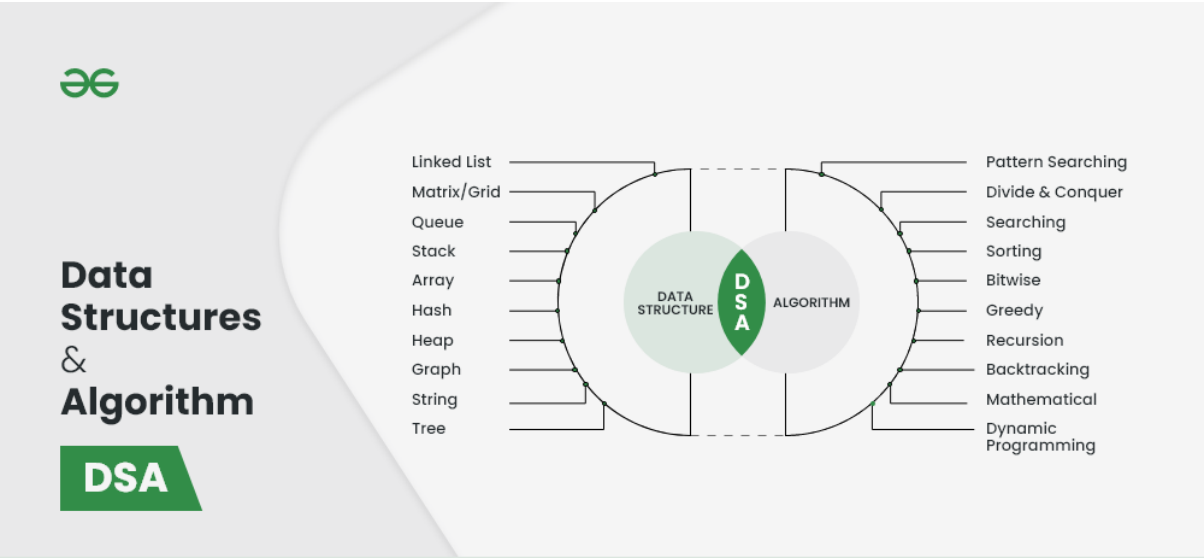


What is Data Structure?

A data structure is defined as a particular way of storing and organizing data in our devices to use the data efficiently and effectively. The main idea behind using data structures is to minimize the time and space complexities. An efficient data structure takes minimum memory space and requires minimum time to execute the data.

What is Algorithm?

Algorithm is defined as a process or set of well-defined instructions that are typically used to solve a particular group of problems or perform a specific type of calculation. To explain in simpler terms, it is a set of operations performed in a step-by-step manner to execute a task.

1. Learn about Complexities

Here comes one of the interesting and important topics. The primary motive to use DSA is to solve a problem effectively and efficiently. How can you decide if a program written by you is efficient or not? This is measured by complexities. Complexity is of two types:

1. Time Complexity: **Time complexity is used to measure the amount of time required to execute the code.**
2. Space Complexity: **Space complexity means the amount of space required to execute successfully the functionalities of the code.**

   You will also come across the term **Auxiliary Space** very commonly in DSA, which refers to the extra space used in the program other than the input data structure.

Both of the above complexities are measured with respect to the input parameters. But here arises a problem. The time required for executing a code depends on several factors, such as:

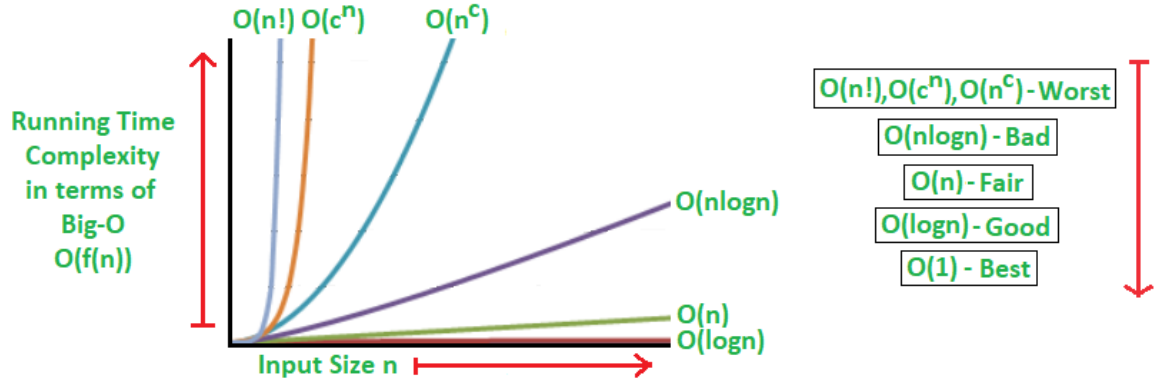
- **The number of operations performed in the program,**
- The speed of the device, and also
- The speed of data transfer if being executed on an online platform.

So how can we determine which one is efficient? The answer is the use of asymptotic notation.

Asymptotic notation *is a* ***mathematical tool that calculates the required time in terms of input size and does not require the execution of the code.***

It neglects the system-dependent constants and is related to only the number of modular operations being performed in the whole program. The following 3 asymptotic notations are mostly used to represent the time complexity of algorithms:

- **Big-O Notation (O)** – Big-O notation specifically describes the worst-case scenario.
- **Omega Notation (Ω)** – Omega(Ω) notation specifically describes the best-case scenario.
- **Theta Notation (θ)** – This notation represents the average complexity of an algorithm.



Rate of Growth of Algorithms

The most used notation in the analysis of a code is the **Big O Notation** which gives an upper bound of the running time of the code (or the amount of memory used in terms of input size).
To learn about complexity analysis in detail, you can refer to our complete set of articles on the **Analysis of Algorithms**.

2. Learn Data Structures

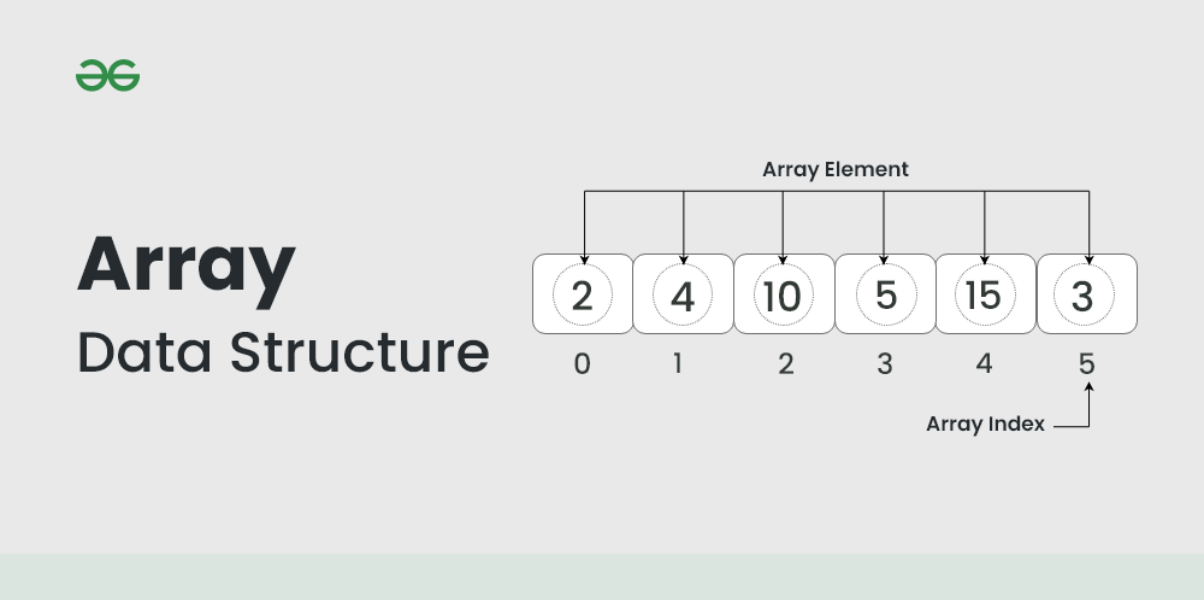
Here comes the most crucial and the most awaited stage of the roadmap for learning data structure and algorithm – the stage where you start learning about DSA. The topic of DSA consists of two parts:

- **Data Structures**
- **Algorithms**

Though they are two different things, they are highly interrelated, and it is very important to follow the right track to learn them most efficiently.

1. Array

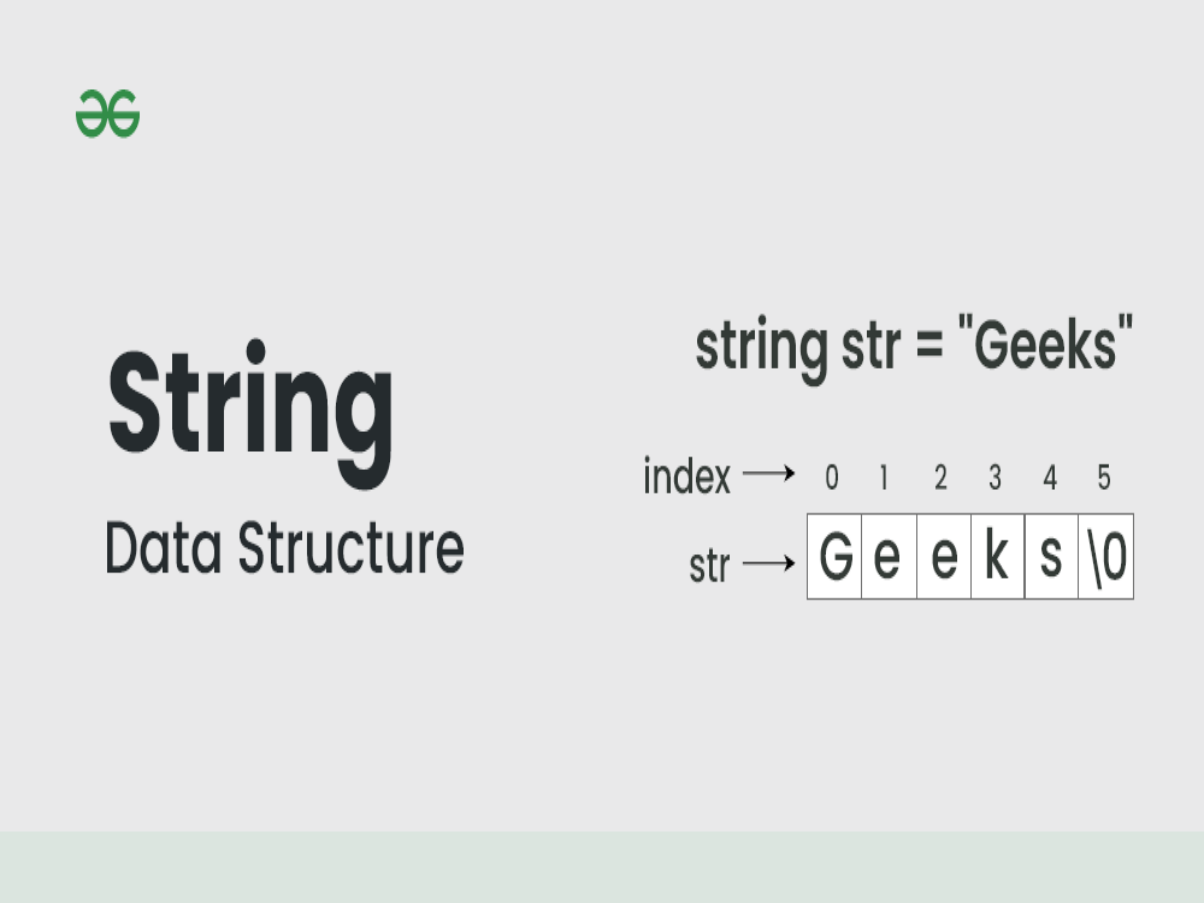
The most basic yet important data structure is the array. It is a linear data structure. An array is a collection of homogeneous data types where the elements are allocated contiguous memory. Because of the contiguous allocation of memory, any element of an array can be accessed in constant time. Each array element has a corresponding index number.



Array Data Structure

2. String

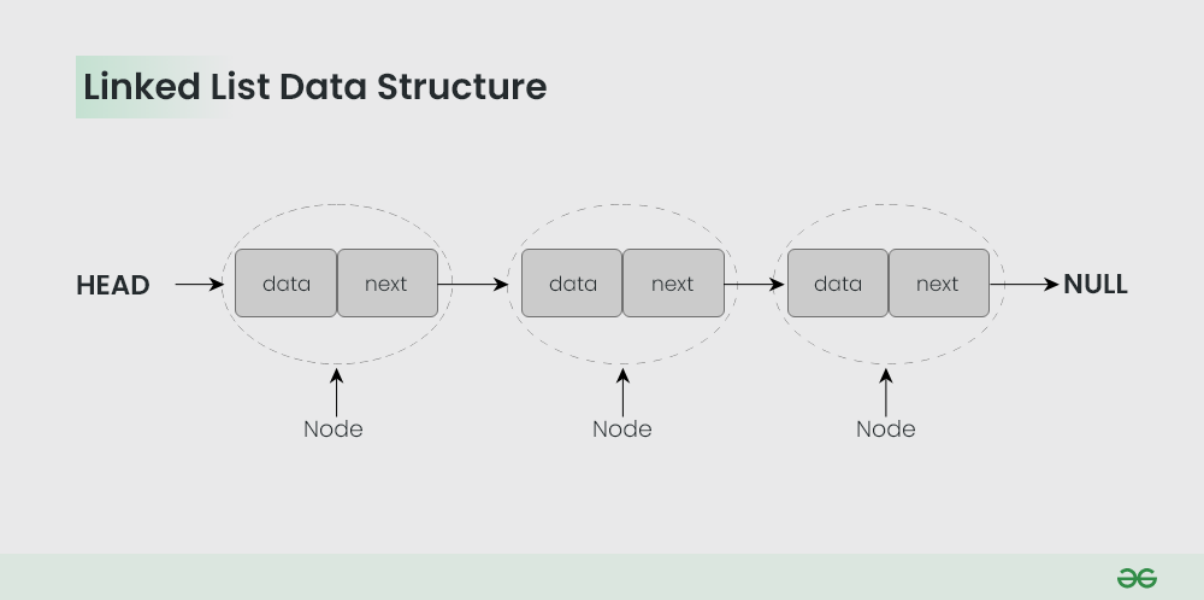
A string is also a type of array. It can be interpreted as an array of characters. **But it has some special characteristics like the last character of a string is a null character to denote the end of the string**. Also, there are some unique operations, **like concatenation which concatenates two strings into one**.



String Data Structure

3. Linked Lists

As the above data structures, **the linked list is also a linear data structure**. But Linked List is different from Array in **its configuration. It is not allocated to contiguous memory locations**. Instead, each node of the linked list is **allocated to some random memory space and the previous node maintains a pointer that points to this node.** So, **no direct memory access of any node is possible** and it is also dynamic i.e., the **size of the linked list can be adjusted at any time.**



Linked List Data Structure

The topics which you must want to cover are:

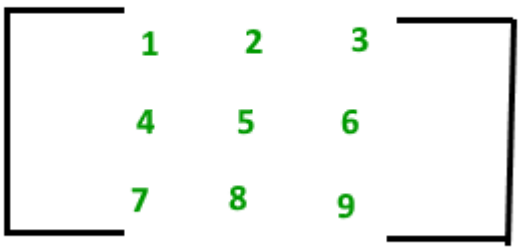
- Singly Linked List – In this, each node of the linked list points only to its next node.
- Circular Linked List – This is the type of linked list where the last node points back to the head of the linked list.
- Doubly Linked List – In this case, each node of the linked list holds two pointers, one point to the next node and the other points to the previous node.

4. Matrix/Grid

A matrix represents a collection of numbers arranged in an order of rows and columns. It is necessary to enclose the elements of a matrix in parentheses or brackets.

For example:

A matrix with 9 elements is shown below.

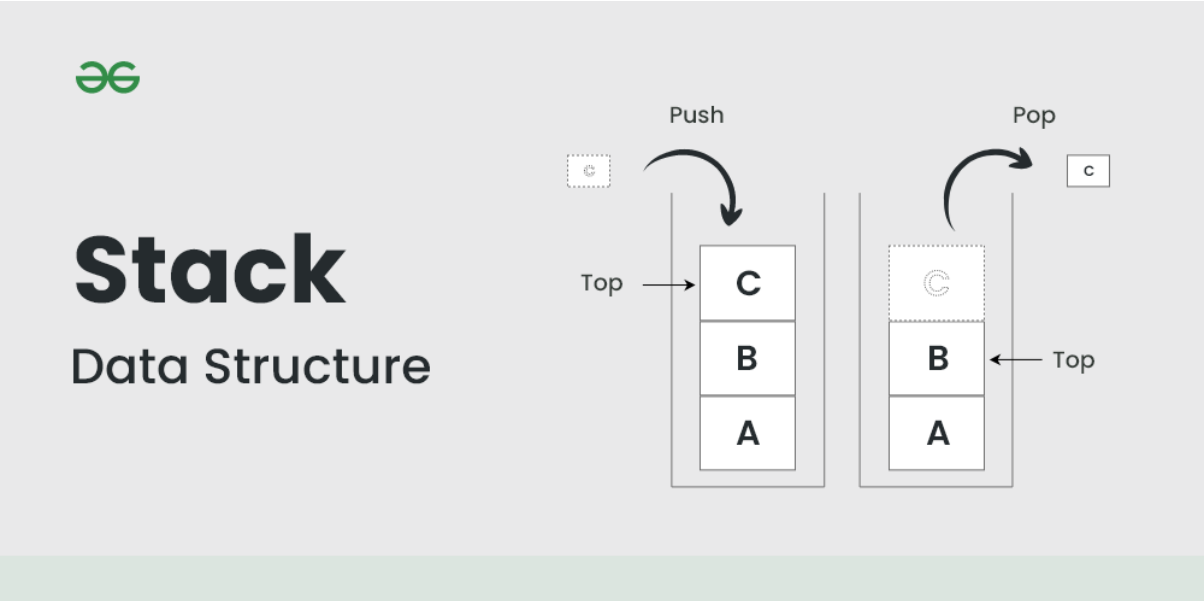


This Matrix **M** has 3 rows and 3 columns. Each element of matrix **M** can be referred to by its row and column number. For example, **M[2][3]** = 6.

5. Stack

Now you should move to some more complex data structures, such as Stack and Queue.

Stack *is a linear data structure which follows a particular order in which the operations are performed. The order may be* ***LIFO(Last In First Out) or FILO(First In Last Out)***.



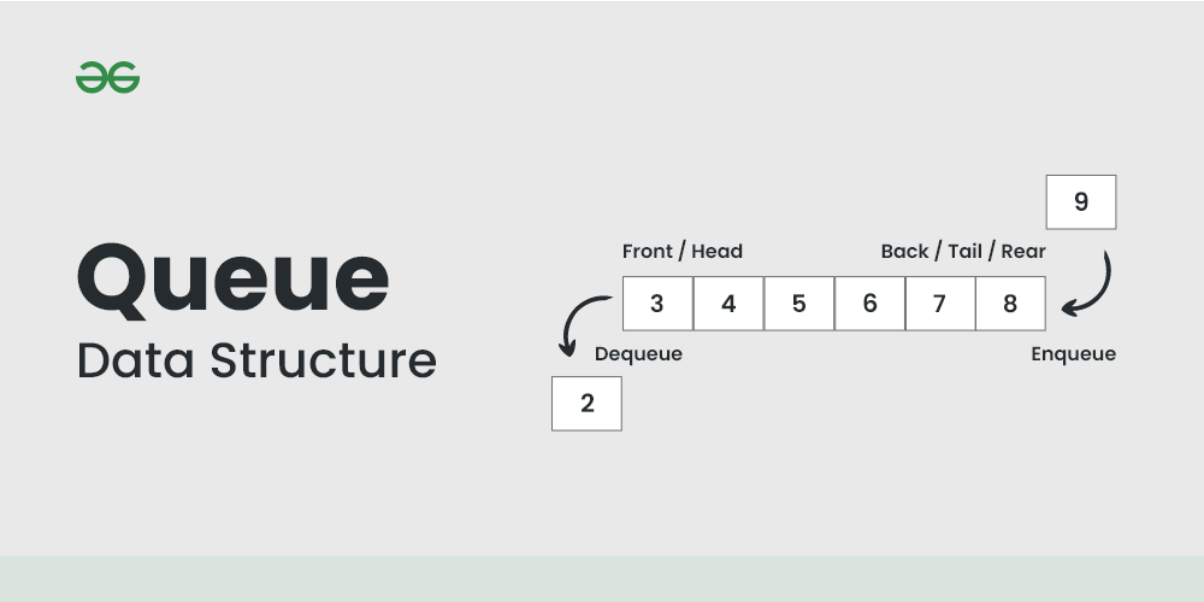
Stack Data Structure

The reason why Stack is considered a complex data structure is that it uses other data structures for implementation, such as Arrays, Linked lists, etc. based on the characteristics and features of Stack data structure.

6. Queue

Another data structure that is similar to Stack, yet different in its characteristics, is Queue.

*A **Queue** is a linear structure which follows **First In First Out (FIFO)** approach in its individual operations.*



Queue Data Structure

A queue can be of different types like

- Circular queue – In a circular queue the last element is connected to the first element of the queue

- Double-ended queue (or known as deque) – A double-ended queue is a special type of queue where one can perform the operations from both ends of the queue.
- Priority queue – It is a special type of queue where the elements are arranged as per their priority. A low priority element is dequeued after a high priority element.

7. Heap

*A Heap is a special **Tree-based Data Structure** in which the tree is a **complete binary tree**.*

Types of heaps:

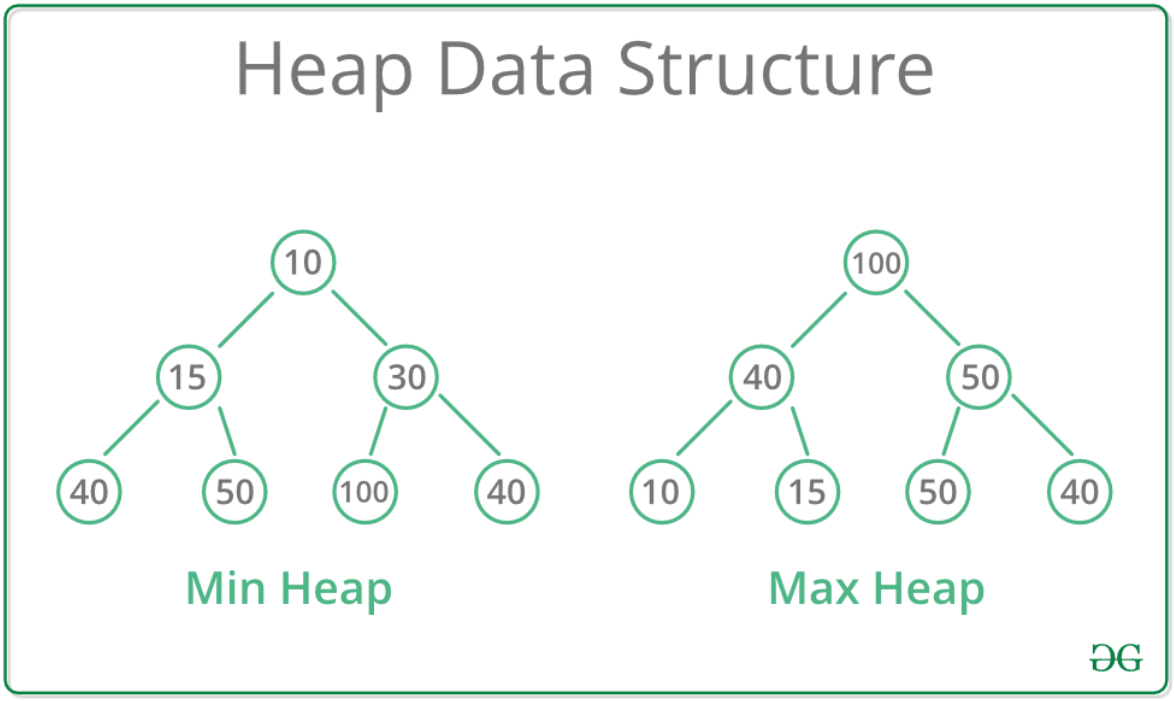
Generally, heaps are of two types.

Max-Heap:

In this heap, the value of the root node must be the greatest among all its child nodes and the same thing must be done for its left and right sub-tree also.

Min-Heap:

In this heap, the value of the root node must be the smallest among all its child nodes and the same thing must be done for its left ans right sub-tree also.



Types of Heap Data Structure

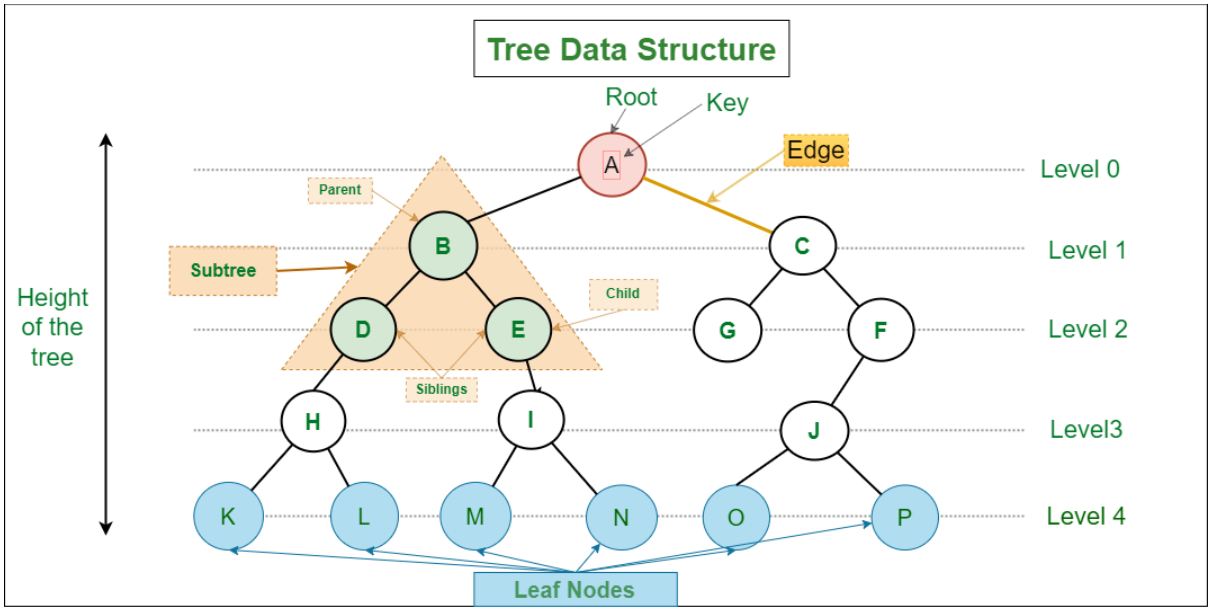
8. Hash

Hashing** refers **to the process of generating a fixed-size output from an input of variable size using the mathematical formulas known as hash functions**. This technique **determines an index or location for the storage of an item in a data structure.

9. Tree Data Structures

After having the basics covered about the linear data structure, now it is time to take a step forward to learn about the **non-linear data structures.** The first non-linear data structure you should learn is the tree.

Tree data structure *is similar* ***to a tree we see in nature but it is upside down****. It also has a root and leaves. The root is the first node of the tree and the leaves are the ones at the bottom-most level. The special characteristic of a tree is that there is only one path to go from any of its nodes to any other node.*



Tree Data Structure

Based on the maximum number of children of a node of the tree it can be –

- Binary tree – This is a special type of tree where each node can have a maximum of 2 children.
- Ternary tree – This is a special type of tree where each node can have a maximum of 3 children.
- N-ary tree – In this type of tree, a node can have at most N children.

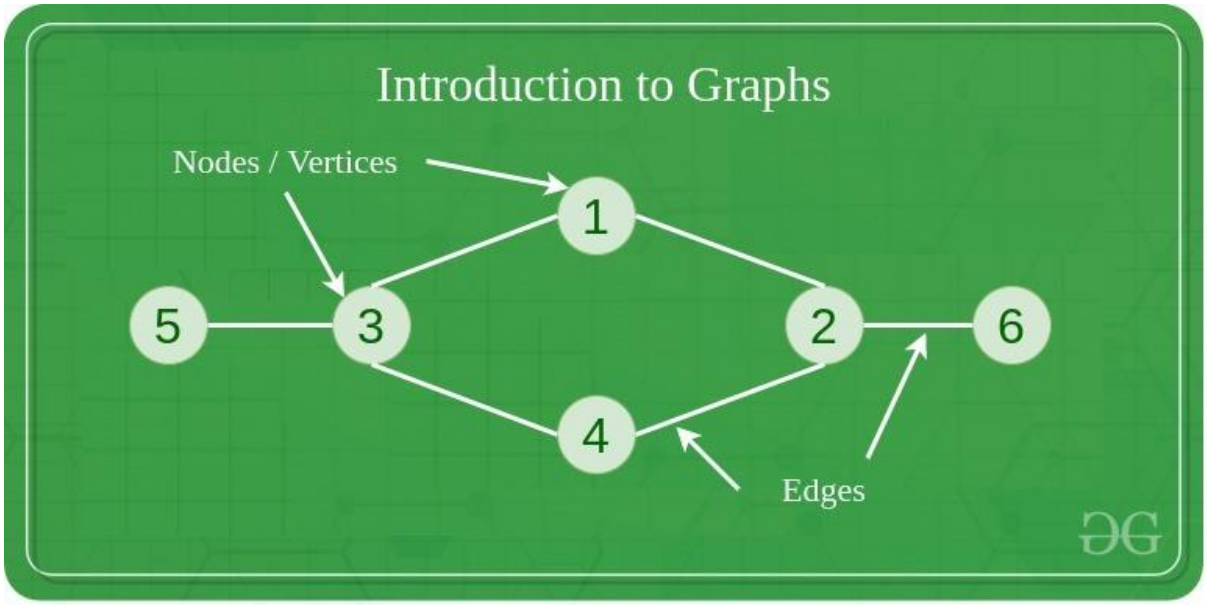
Based on the configuration of nodes there are also several classifications. Some of them are:

- Complete Binary Tree – In this type of binary tree all the levels are filled except maybe for the last level. But the last level elements are filled as left as possible.
- Perfect Binary Tree – A perfect binary tree has all the levels filled
- Binary Search Tree – A binary search tree is a special type of binary tree where the smaller node is put to the left of a node and a higher value node is put to the right of a node
- Ternary Search Tree – It is similar to a binary search tree, except for the fact that here one element can have at most 3 children.

10. Graph Data Structure

Another important **non-linear data structure is the graph.** It is similar to the Tree data structure, with the difference that **there is no particular root or leaf node, and it can be traversed in any order.**

*A **Graph** is a non-linear data structure consisting of a finite set of vertices (or nodes) and a set of edges that connect a pair of nodes.*



Graph Data Structure

Each edge shows a connection between a pair of nodes. This data structure helps solve many real-life problems. Based on the orientation of the edges and the nodes there are various types of graphs.

Here are some must to know concepts of graphs:

- Types of graphs – There are different types of graphs based on connectivity or weights of nodes.
- Introduction to BFS and DFS – These are the algorithms for traversing through a graph
- Cycles in a graph – Cycles are a series of connections following which we will be moving in a loop.
- Topological sorting in the graph
- Minimum Spanning tree in graph

3. Learn Algorithms

Once you have cleared the concepts of Data Structures, now it's time to start your journey through the Algorithms. Based on the type of nature and usage, the Algorithms are grouped together into several categories, as shown below:

1. Searching Algorithm

Now we have learned about some linear data structures and is time to learn about some basic and most used algorithms which are hugely used in these types of data structures. One such algorithm is the searching algorithm.

Searching algorithms *are used to find a specific element in an array, string, linked list, or some other data structure.*

The most common searching algorithms are:

- Linear Search – In this searching algorithm, we check for the element iteratively from one end to the other.
- Binary Search – In this type of searching algorithm, we break the data structure into two equal parts and try to decide in which half we need to find for the element.
- Ternary Search – In this case, the array is divided into three parts, and based on the values at partitioning positions we decide the segment where we need to find the required element.

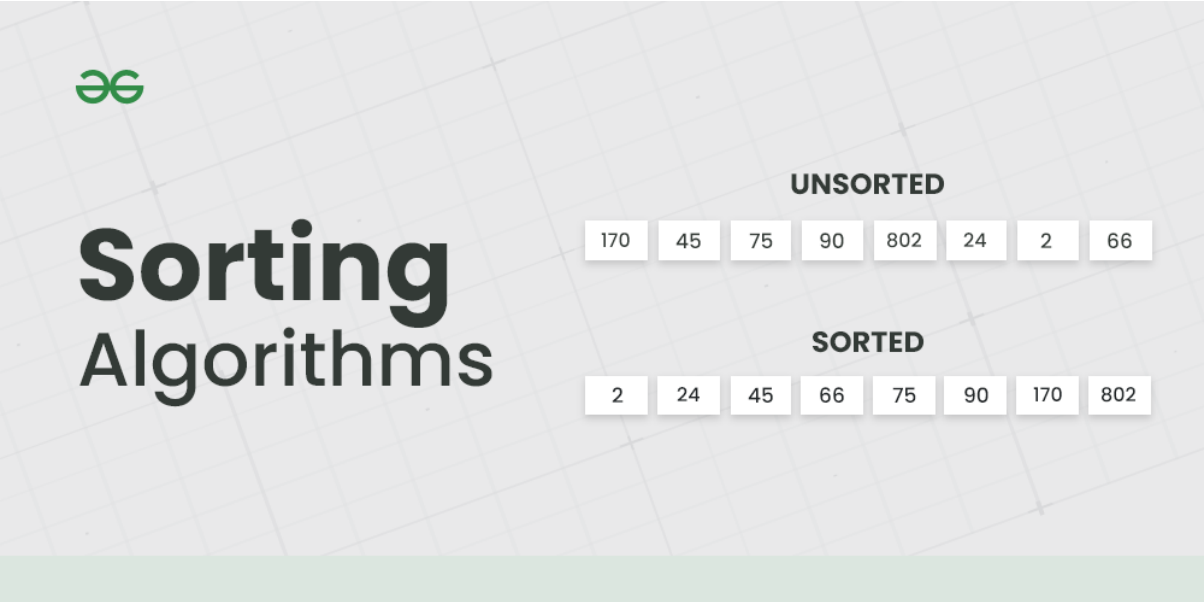
Besides these, there are other searching algorithms also like

- Jump Search
- Interpolation Search
- Exponential Search

2. Sorting Algorithm

Here is one other most used algorithm. Often we need to arrange or sort data as per a specific condition. The sorting algorithm is the one that is used in these cases. **Based on conditions we can sort a set of homogeneous data in order like sorting an array in increasing or decreasing order.**

Sorting Algorithm *is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.*



An example to show Sorting

There are a lot of different types of sorting algorithms. Some widely used algorithms are:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort

There are several other sorting algorithms also and they are beneficial in different cases. You can learn about them and more in our dedicated article on Sorting algorithms.

3. Divide and Conquer Algorithm

This is one interesting and important algorithm to be learned in your path of programming. As the name suggests, **it breaks the problem into parts, then solves each part and after that again merges the solved subtasks to get the actual problem solved.**

Divide and Conquer *is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.*

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

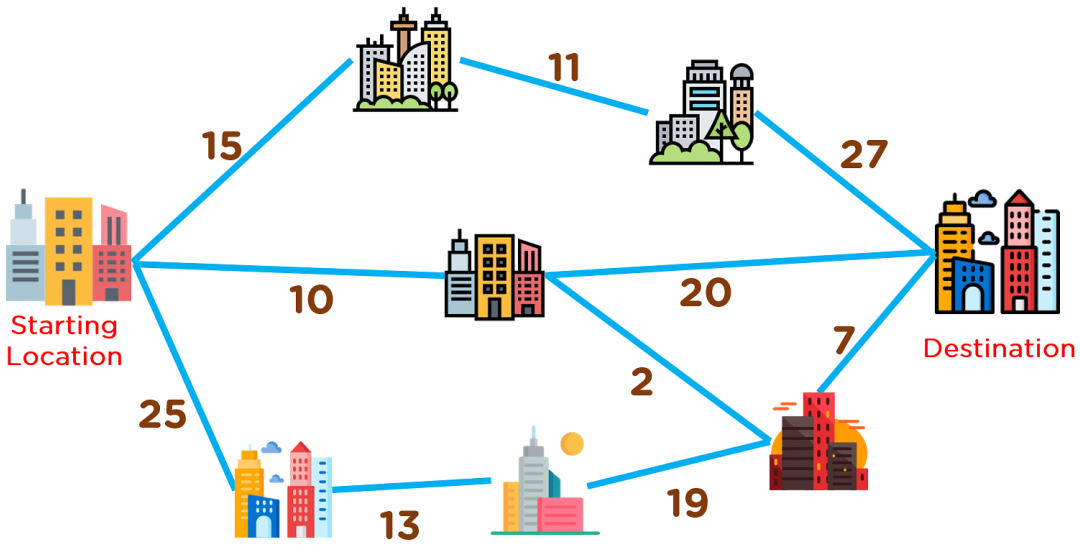
This is the primary technique mentioned in the two sorting algorithms ***Merge Sort*** **and** ***Quick Sort*** **which are mentioned earlier.** To learn more about the technique, the cases where it is used, and its implementation and solve some interesting problems, please refer to the dedicated article Divide and Conquer Algorithm.

4. Greedy Algorithms

The greedy method is **one of the strategies like Divide and conquer used to solve the problems**. This method is **used for solving optimization problems.** An optimization problem **is a problem that demands either maximum or minimum results.**

The Greedy method is the simplest and straightforward approach. It is not an algorithm, but it is a technique. The main function of this approach **is that the decision is taken on the basis of the currently available information**. Whatever the current information is present, the decision is made **without worrying about the effect of the current decision in future.**

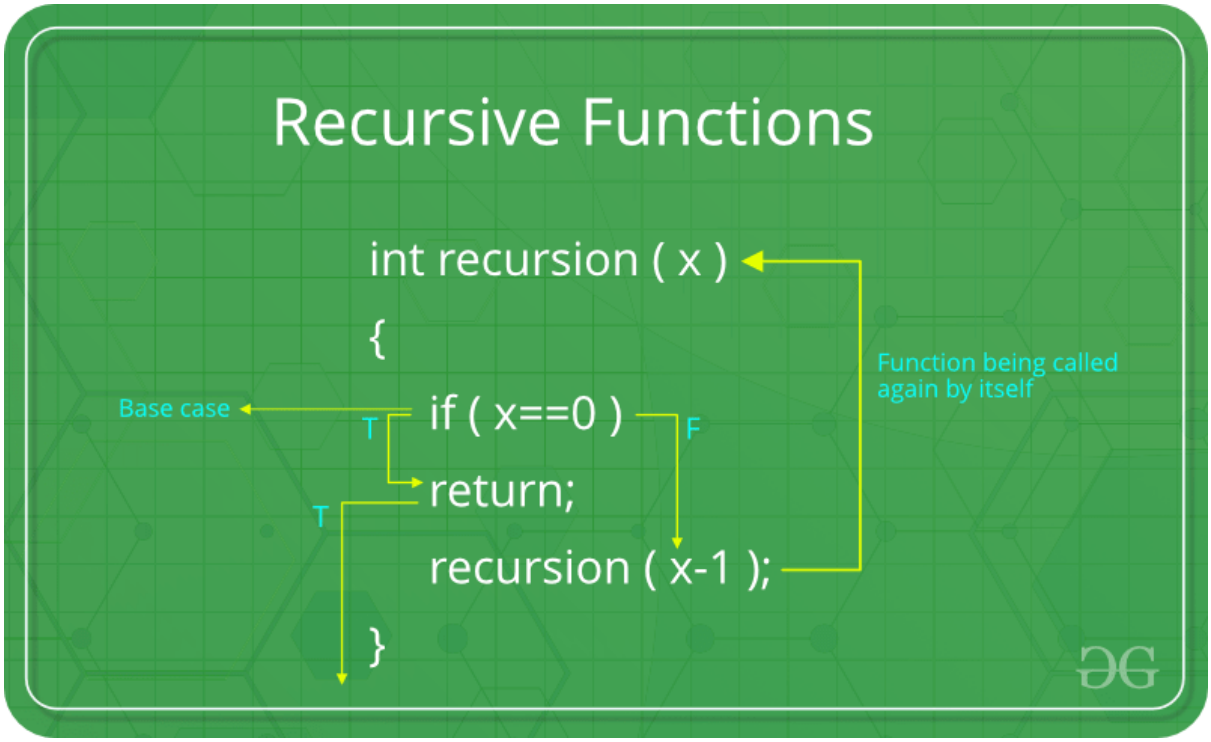
This technique is basically used to determine the **feasible solution that may or may not be optimal.** The feasible solution **is a subset that satisfies the given criteria.** The optimal solution is the solution **which is the best and the most favourable solution in the subset.** In the case of feasible, **if more than one solution satisfies the given criteria then those solutions will be considered as the feasible**, whereas the **optimal solution is the best solution among all the solutions.**



Example of Greedy Algorithm

5. Recursion

Recursion is one of the most important algorithms **which uses the concept of code reusability and repeated usage of the same piece of code.**



Recursion

The point which makes Recursion one of the most used algorithms is that it forms the base for many other algorithms such as:

- Tree traversals
- Graph traversals
- Divide and Conquers Algorithms
- Backtracking algorithms

6. Backtracking Algorithm

As mentioned earlier, the **Backtracking algorithm is derived from the Recursion algorithm, with the option to revert if a recursive solution fails**, i.e., in case a solution fails, the program traces back to the moment where it failed and builds on another solution.

So basically, it tries out all the possible solutions and finds the correct one.

Backtracking *is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time*

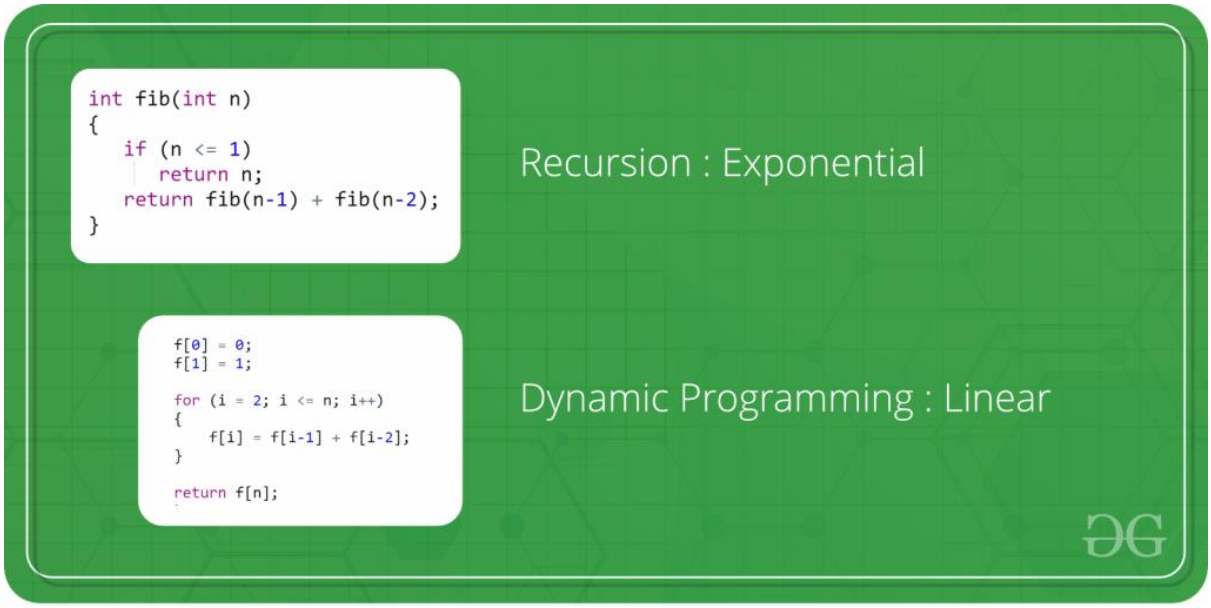
Some important and most common problems of backtracking algorithms, that you must solve before moving ahead, are:

- N-Queen problem

7. Dynamic Programming

Another crucial algorithm is dynamic programming. **Dynamic Programming is mainly an optimization over plain recursion.** Wherever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using Dynamic Programming.

The main concept of the ***Dynamic Programming algorithm*** *is to use the previously calculated result to avoid repeated calculations of the same subtask which helps in reducing the time complexity.*



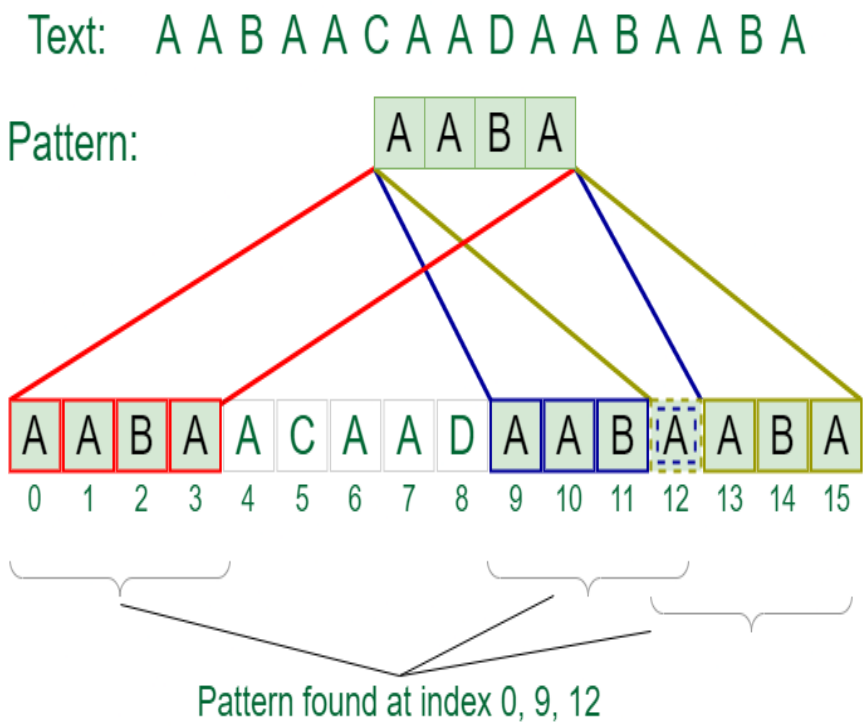
Dynamic Programming

To learn more about dynamic programming and practice some interesting problems related to it, refer to the following articles:

- Tabulation vs Memoization
- Overlapping Subproblems Property

8. Pattern Searching

The Pattern Searching algorithms are sometimes also referred to as **String Searching Algorithms** and are considered as a part of the **String algorithms**. These algorithms are useful in the case of searching a string within another string.



9. Mathematical Algorithms

These algorithms are designed to solve Mathematical and Number Theory problems. They require in-depth knowledge of different mathematical concepts like

- LCM
- Prime Factorization and Divisors
- Fibonacci Numbers
- Set Theory
- Factorial
- Prime numbers

10. Geometric Algorithms

These algorithms are designed to solve Geometric Problems. They require in-depth knowledge of different mathematical concepts like:

☐ Lines
☐ Triangle
☐ Rectangle

- ☐ Square
- ☐ Circle
- ☐ 3D Objects
- ☐ Quadrilateral
- ☐ Polygon & Convex Hull

11. Bitwise Algorithms

The **Bitwise Algorithms** is used to perform operations at the bit-level or to manipulate bits in different ways. The bitwise operations are found to be much faster and are sometimes used to improve the efficiency of a program.

For example: To check if a number is even or odd. This can be easily done by using Bitwise-AND(&) operator. If the last bit of the operator is set than it is ODD otherwise it is EVEN. Therefore, if **num & 1** not equals to zero than num is ODD otherwise it is EVEN.

12. Randomized Algorithms

An algorithm **that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm.** For example, in Randomized Quick Sort, we use a random number to pick the next pivot (or we randomly shuffle the array). Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms.

13. Branch and Bound Algorithm

Branch and bound is an algorithm design paradigm which is generally **used for solving combinatorial optimization problems.** These problems are **typically exponential in terms of time complexity** and may require exploring **all possible permutations** in **worst case**. The Branch and Bound Algorithm technique solves these problems relatively quickly.