**Data structures**

It is an essential component that help to organize and store data efficiently in a computer memory. They provide a way to manage and manipulate data effectively, enabling faster access, insertion, and deletion.

**Algorithm**

It is a process or set of steps to accomplish a certain task. It is the foundation for being a successful problem solver and a developer.

**Approach to solve a problem**

1. Devise a plan for solving a problem
2. Master common problem solving patterns

**How to solve problems**

1. Understand the problem
2. Explore concrete examples
3. Break it down
4. Solve/Simplify
5. Look back and refactor

**Questions to ask while refactoring**

1. Can you check the result?
2. Can you derive the result differently?
3. Can you understand it at a glance?
4. Can you use the result or method for some other problem?
5. Can you improve the performance of your solution?
6. Can you think of other ways to refactor?
7. Check how other people have solved this problem?

**Algorithm Analysis**

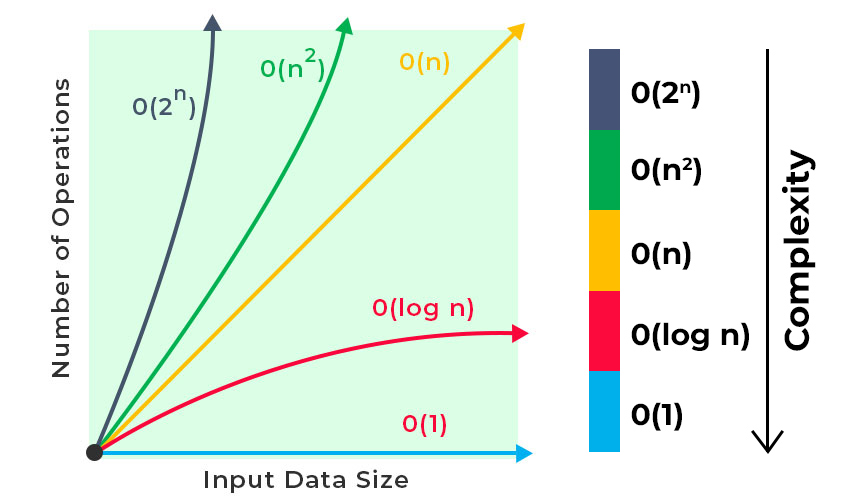
Algorithm analysis evaluates how an algorithm’s runtime or resource consumption grows as the input size increases. This analysis helps us compare and choose algorithms based on their efficiency. Five common notations are used for algorithm analysis:

* Big O (O): Upper Bound — represents the worst-case scenario.
* Big Theta (Θ): Tight Bound — represents upper and lower bounds.
* Big Omega (Ω): Lower Bound — represents the best-case scenario.
* Small Theta (θ): Asymptotically Tight Bound — Like Big Theta but with a more precise constant factor.
* Small O (o): Upper Bound (excluding equal) — Represents an upper bound strictly greater than.

Time and space complexity measured depends on algorithm, not hardware used to run it.

**Dominant term**

It is the term that grows the fastest. Ex- n2 + 5n + 6. Therefore the time complexity is O(n2)



Logarithmic time complexity is good.

**Space Complexity**Space complexity is the total space taken by the algorithm with respect to the input size plus the auxiliary space that the algorithm uses. **Auxiliary Space Complexity**

The extra space that is taken by an algorithm temporarily to finish its work, excluding the space taken up by the inputs.

**Space complexity rules of thumb**

1. Primitive data types (Boolean, Number, Undefined, Null) takes constant space.
2. Strings, Reference types (Array, Object) takes O(n) space.

**Big O of objects**

* Insertion, Removal, Access - O(1)
* Searching - O(n)

**Few common problem solving patterns**

* Frequency counter
* Multiple pointers
* Sliding window
* Divide and Conquer
* Dynamic Programming
* Greedy Algorithm
* Backtracking
* And many more….