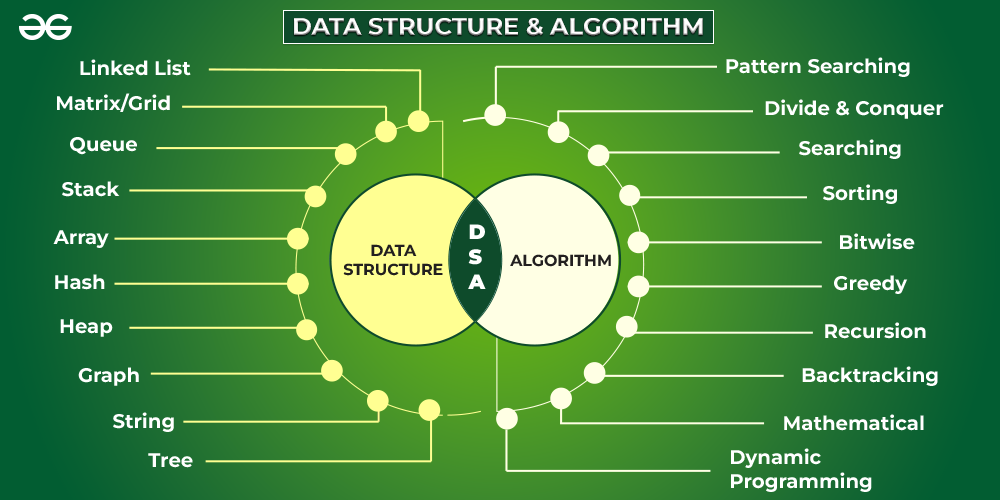
**Algorithm**



A  set of rules to be followed in calculations or other problem-solving operations ” Or ” A procedure for solving a mathematical problem in a finite number of steps that frequently by recursive operations

# **Types of Algorithms :**

## **Brute Force:**

Brute force algorithm is a technique that guarantees solutions for problems of any domain helps in solving the simpler problems and also provides a solution that can serve as a benchmark for evaluating other design techniques but takes a lot of run time and inefficient.

PROS AND CONS OF BRUTE FORCE ALGORITHM:

**Pros:**

1. The brute force approach is a guaranteed way to find the correct solution by listing all the possible candidate solutions for the problem.
2. It is a generic method and not limited to any specific domain of problems.
3. The brute force method is ideal for solving small and simpler problems.
4. It is known for its simplicity and can serve as a comparison benchmark.

**Cons:**

1. The brute force approach is inefficient. For real-time problems, algorithm analysis often goes above the O(N!) order of growth.
2. This method relies more on compromising the power of a computer system for solving a problem than on a good algorithm design.
3. Brute force algorithms are slow.
4. Brute force algorithms are not constructive or creative compared to algorithms that are constructed using some other design paradigms

*If there is a lock of 4-digit PIN. The digits to be chosen from 0-9 then the brute force will be trying all possible combinations one by one like 0001, 0002, 0003, 0004, and so on until we get the right PIN. In the worst case, it will take 10,000 tries to find the right combination.*

**The most common searching algorithms are:**

1. Linear search
2. Binary Search
3. Ternary Search

Other

1. [Jump Search](https://www.geeksforgeeks.org/jump-search/)
2. [Interpolation Search](https://www.geeksforgeeks.org/interpolation-search/)
3. [Exponential Search](https://www.geeksforgeeks.org/exponential-search/)
4. [Sublist Search (Search a linked list in another list)](https://www.geeksforgeeks.org/sublist-search-search-a-linked-list-in-another-list/)
5. [Fibonacci Search](https://www.geeksforgeeks.org/fibonacci-search/)
6. [The Ubiquitous Binary Search](https://www.geeksforgeeks.org/the-ubiquitous-binary-search-set-1/)
7. [Recursive program to linearly search an element in a given array](https://www.geeksforgeeks.org/recursive-c-program-linearly-search-element-given-array/)
8. [Recursive function to do substring search](https://www.geeksforgeeks.org/recursive-function-to-do-substring-search/)
9. [Unbounded Binary Search Example (Find the point where a monotonically increasing function becomes positive first time)](https://www.geeksforgeeks.org/find-the-point-where-a-function-becomes-negative/)

**Searching Algorithm Approach**

1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: [Linear Search](https://www.geeksforgeeks.org/linear-search/).
2. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: [Binary Search](https://www.geeksforgeeks.org/binary-search/).

# **Binary Search**

**Binary Search**. It is also known as half interval search, logarithmic search, and binary chop.  It is a searching algorithm that helps find the element in the array and return the respective index. Additionally, we could also return a flag value to denote the element does not exist.

Binary Search follows the **divide and conquer** policy, i.e., searches an element by repeatedly ***dividing the array into two parts*** for every comparison.

**Note:** It can only be applied on sorted arrays or functions which are monotonic in nature.

Int binarySearch(arr, x, Left, right)

repeat till left = right

mid = (left + right)/2

if (x == arr[mid])

return mid

else if (x > arr[mid]) // x is on the right side

left = mid + 1

else // x is on the left side

right = mid - 1

## **Time and Space Complexity of Binary search**

**Time Complexity**–

1. **Best Time Complexity**: O(1) as if the element is present at the middle index, only one comparison is required.
2. **Average Time Complexity**: It is the average of time complexities to find the element at every index. So, since the array gets divided into half in every iteration, it follows: n -> n/2 -> n/4 -> n/8….. n/(2^k)

So, follows a Recurrence Relation : T(n) = T(n/2) + c (where c is constant)

This relation could be solved using a Recurrence Tree or Master Method, hence giving a complexity of O(log n (base 2)).

1. **Worst Time Complexity**: O(logn (base 2)) due to the recurrence relation.

**Space Complexity**: O(1) as we aren’t utilizing or creating any space apart from the given array length as an input.

### Sorting Algorithm

**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 elements in the respective data structure.

* [Bubble Sort](http://www.geeksforgeeks.org/bubble-sort/)
* [Selection Sort](http://www.geeksforgeeks.org/selection-sort/)
* [Insertion Sort](http://www.geeksforgeeks.org/insertion-sort/)
* [Quick Sort](http://www.geeksforgeeks.org/quick-sort/)
* [Merge Sort](http://www.geeksforgeeks.org/merge-sort/)

# Bubble Sort Algorithm

* Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
* This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

**Complexity of bubble sort**

* Time complexity - O(n^2n2 )
* Space complexity - O(1)