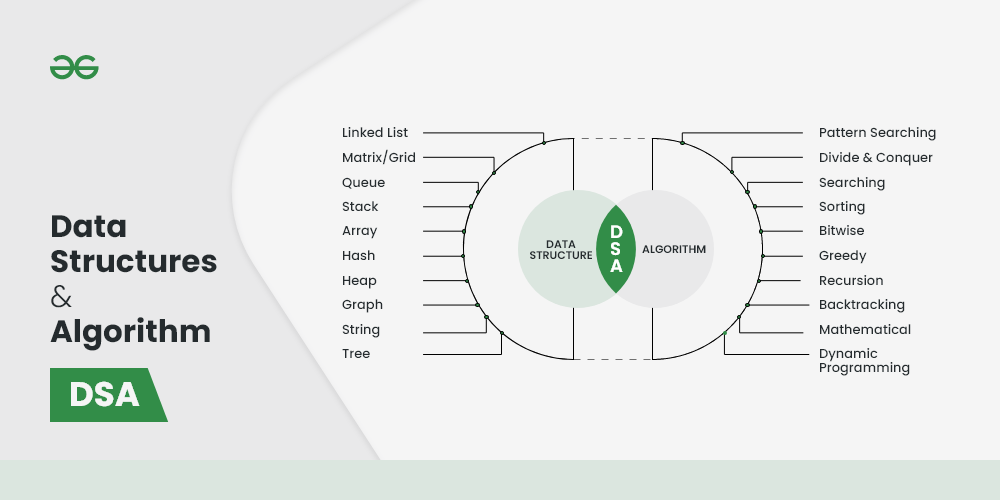
**Algorithms:**



**1. 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](https://www.geeksforgeeks.org/linear-search/) – In this searching algorithm, we check for the element iteratively from one end to the other.

<https://www.geeksforgeeks.org/linear-search/>

* [Binary Search](https://www.geeksforgeeks.org/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.

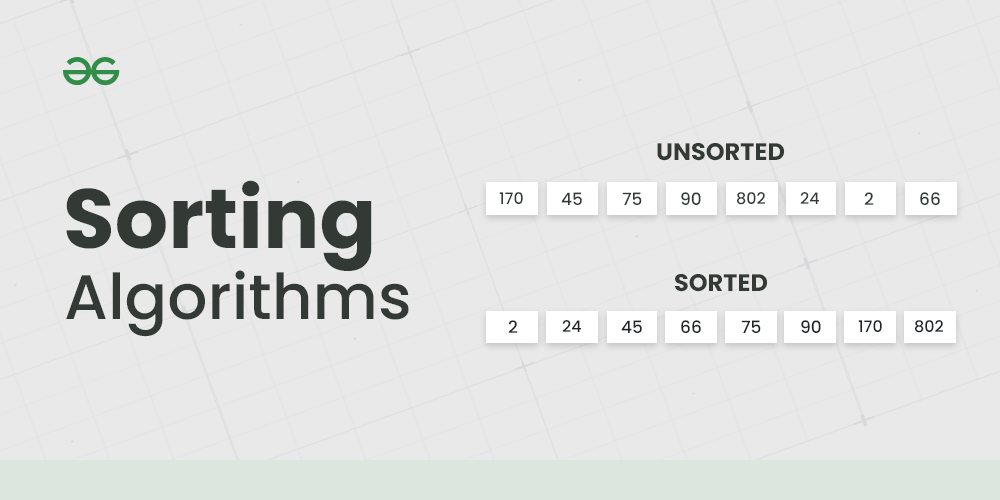
<https://www.geeksforgeeks.org/binary-search/>

* [Ternary Search](https://www.geeksforgeeks.org/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.

<https://www.geeksforgeeks.org/ternary-search/>

### 2. 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 element in the respective data structure.*



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

* [Bubble Sort](http://www.geeksforgeeks.org/bubble-sort/)

**Bubble Sort** is the simplest [sorting algorithm](https://www.geeksforgeeks.org/sorting-algorithms/) that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

<https://www.geeksforgeeks.org/bubble-sort/>

* [**Selection Sort**](http://www.geeksforgeeks.org/selection-sort/)

**Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.**

<https://www.geeksforgeeks.org/selection-sort/>

* [Insertion Sort](http://www.geeksforgeeks.org/insertion-sort/)

***Insertion sort****is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.*

<https://www.geeksforgeeks.org/insertion-sort/>

* [Quick Sort](http://www.geeksforgeeks.org/quick-sort/)

***QuickSort****is a sorting algorithm based on the*[*Divide and Conquer algorithm*](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/)*that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.*

<https://www.geeksforgeeks.org/quick-sort/>

* [Merge Sort](http://www.geeksforgeeks.org/merge-sort/)

***Merge sort****is defined as a*[*sorting algorithm*](https://www.geeksforgeeks.org/sorting-algorithms/)*that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.*

<https://www.geeksforgeeks.org/merge-sort/>

### 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* .

[*https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/*](https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/)

### Greedy Algorithm

### 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. So the problems where choosing locally optimal also leads to global solution are the best fit for Greedy.

### [Greedy Algorithm](http://www.geeksforgeeks.org/greedy-algorithms/) is defined as a method for solving ****optimization problems**** by taking decisions that result in the most evident and immediate benefit irrespective of the final outcome. It works for cases where ****minimization****or ****maximization****leads to the required solution.

### <https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/> 🡪 characteristics, uses and examples .

### Recursion

### The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily. Examples of such problems are [Towers of Hanoi (TOH)](https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/), [Inorder/Preorder/Postorder Tree Traversals](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/), [DFS of Graph](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), etc. A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems.

### <https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/>

### Backtracking Algorithm

### [****Backtracking****](http://www.geeksforgeeks.org/backtracking-algorithms/) 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](https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/)

<https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>

* [m-coloring problem](https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/)

<https://www.geeksforgeeks.org/m-coloring-problem/>

* [Hamiltonian cycle](https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/)

<https://www.geeksforgeeks.org/hamiltonian-cycle/>

* [Sudoku](https://www.geeksforgeeks.org/sudoku-backtracking-7/)

### <https://www.geeksforgeeks.org/sudoku-backtracking-7/>

### Dynamic Programming

### *The main concept of the*[*Dynamic Programming algorithm*](https://www.geeksforgeeks.org/dynamic-programming/)*is to use the previously calculated result to avoid repeated calculations of the same subtask which helps in reducing the time complexity.*

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

* [Tabulation vs Memoization](https://www.geeksforgeeks.org/tabulation-vs-memoizatation/)
* [Optimal Substructure Property](https://www.geeksforgeeks.org/dynamic-programming-set-2-optimal-substructure-property/)
* [Overlapping Subproblems Property](https://www.geeksforgeeks.org/dynamic-programming-set-1/)
* [How to solve a Dynamic Programming Problem?](https://www.geeksforgeeks.org/solve-dynamic-programming-problem/)
* [Bitmasking and Dynamic Programming | Set 1](https://www.geeksforgeeks.org/bitmasking-and-dynamic-programming-set-1-count-ways-to-assign-unique-cap-to-every-person/)
* [Bitmasking and Dynamic Programming | Set-2 (TSP)](https://www.geeksforgeeks.org/bitmasking-dynamic-programming-set-2-tsp/)
* [Digit DP | Introduction](https://www.geeksforgeeks.org/digit-dp-introduction/)