**Brute Force Algorithms:**

* An algorithm is a well-defined procedure that allows a computer to solve a problem.
* Brute force is a straightforward approach to solving a problem
* a brute-force attack involves systematically checking all possible [keys](https://en.wikipedia.org/wiki/Key_(cryptography)) until the correct key is found
* Brute force is a type of [algorithm](https://simplicable.com/new/algorithm-definition) that tries a large number of patterns to solve a problem. In some cases, they are extremely simple and rely on raw computing power to achieve results.
* It is a very general problem-solving technique.
* A common example of a brute force algorithm is a security threat that attempts to guess a password using known common passwords. Such an algorithm might also try dictionary words or even every combination of ASCII strings of a certain length.
* The time complexity of this searching phase is *O*(*mn*). The expected number of text character comparisons is 2*n*.
* A brute force attack is a trial-and-error method used to obtain information such as a user password or personal identification number (PIN).
* In a brute force attack, automated software is used to generate a large number of consecutive guesses as to the value of the desired data.

Main features:

* no preprocessing phase;
* constant extra space needed;
* always shifts the window by exactly 1 position to the right;
* comparisons can be done in any order;
* searching phase in *O*(*mn*) time complexity;
* 2*n* expected text characters comparisons.

Advantages:

* Chance of actually finding the password. This method is used by default to solve some problems such as sorting, searching, matrix multiplication, binomial expansion etc.

Disadvantage:

* The main disadvantage of the brute-force method is that, for many real-world problems, the number of inputs are to large
* The brute force is not suitable for large number of inputs. Because, crack that password is to difficult if it has multiple number of words, numbers and special characters. Large number of combinations take so much time to find that particular pair.
* Brute force attacks try as many possible answers as possible, this takes a lot of processing power

**Greedy Algorithms:**

* A greedy algorithm is a mathematical process that looks for simple, easy-to-implement solutions to complex, multi-step problems by deciding which next step will provide the most obvious benefit.
* Such algorithms are called greedy because while the optimal solution to each smaller instance will provide an immediate output, the algorithm doesn’t consider the larger problem as a whole.
* Greedy algorithms appear in network [routing](https://en.wikipedia.org/wiki/Routing) as well. Using greedy routing, a message is forwarded to the neighboring node which is "closest" to the destination.
* The greedy method is quite powerful and works well for a wide range of problems. Many algorithms can be viewed as applications of the Greedy algorithms,

such as:

1. [Minimum Spanning Tree](https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/)
2. [Dijkstra’s algorithm for shortest paths from a single source](https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/)
3. [Huffman codes ( data-compression codes )](https://en.wikipedia.org/wiki/Huffman_coding)

* It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

There are a few variations to the greedy algorithm:

* Pure greedy algorithms
* Orthogonal greedy algorithms
* Relaxed greedy algorithms

Greedy algorithms have some advantages and disadvantages:

1. It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem.
2. Analyzing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer).
3. The difficult part is that for greedy algorithms you have to work much harder to understand correctness issues. Even with the correct algorithm, it is hard to prove why it is correct.

**Divide and Conquer Algorithms:**

* Divide and conquer is an algorithm design paradigm based on multi-branched recursion.
* A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
* This divide and conquer technique is the basis of efficient algorithms for all kinds of problems, such as [sorting](https://en.wikipedia.org/wiki/Sorting_algorithm) (e.g., [quicksort](https://en.wikipedia.org/wiki/Quicksort" \o "Quicksort), [merge sort](https://en.wikipedia.org/wiki/Merge_sort)), [multiplying large numbers](https://en.wikipedia.org/wiki/Multiplication_algorithm)(e.g. the [Karatsuba algorithm](https://en.wikipedia.org/wiki/Karatsuba_algorithm" \o "Karatsuba algorithm)).

standard algorithms that are followed Divide and Conquer algorithms:

1) Binary Search is a searching algorithm

2) Quicksort is a sorting algorithm

3) Merge Sort is also a sorting algorithm

4) Closest Pair of Points the problem is to find the closest pair of points in a set of points.

Advantages:

* Solving difficult problems
* Parallelism
* Memory access
* Algorithm efficiency

**Dynamic Programming Algorithms:**

* Dynamic programming is both a mathematical optimization method and a computer programming method.
* DP is simply solving a problem recursively along with storing the calculated information for further usage.
* At every stage in dynamic programming the decision rule is determined by evaluating an objective function called recursive equation.
* Dynamic Programming Examples:

1. Minimum cost from Sydney to Perth

2. Economic Feasibility Study

3. 0/1 Knapsack problem

4. Sequence Alignment problem.

Advantage of dynamic programming:

* The process of breaking down a complex problem into a series of interrelated sub problems often provides insight into the nature of problem
* It has flexibility that allows application access to other types of mathematical programming problems
* Dynamic programming achieves computational savings over complete enumeration.
* Saves time-You are not calculating the obtained results again and again.
* Saves space-You are overwriting the updated values.

Disadvantage of DP:

* It requires a lot of practise to solve the problems in one go.
* It restricts computer codes necessary for inexpensive and widespread use
* The biggest problem is dimensionality. This problems occurs when a particular application is characterized by multiple states.
* It is difficult to develop code using dynamic programming as opposed to greedy technique. Also different DP have different methods of solving them.

**Backtracking Algorithms:**

* Backtracking is nothing but the modified process of the brute force approach. where the technique systematically searches for a solution to a problem among all available options.
* The Algorithmic Approach – Backtracking systematically try and search possibilities to find the solution
* problems like crossword, Sudoku and many other puzzles.

Advantages:

* Comparison with the Dynamic Programming, Backtracking Approach is more effective in some cases.
* Backtracking Algorithm is the best option for solving tactical problem and it is effective for constraint satisfaction problem.
* In greedy Algorithm, getting the Global Optimal Solution is a long procedure and depends on user statements but in Backtracking It Can Easily getable. Backtracking technique is simple to implement and easy to code.

Disadvantages :

* Backtracking Approach is not efficient for solving strategic Problem.
* The overall runtime of Backtracking Algorithm is normally slow.
* To solve Large Problem Sometime it needs to take the help of other techniques like Branch and bound.
* Need Large amount of memory space for storing different state function in the stack for big problem.
* Thrashing is one of the main problem of Backtracking.

Application of Backtracking:

* Optimization and tactical problems
* Constraints Satisfaction Problem
* Solving puzzles and path.

Example:

Sudoku Puzzle :

Backtracking Approach the Algorithm Will Check Each Box’s value if the value is in the same row, or same column or same sub-square box. If not, then it placed the value in the box. And go to the next box for the next value, and check the Above Condition. If there is any duplicate value, then The Algorithm check for the next value.