**BRUTEFORCE ALGORITHM:**

* The brute force algorithm consists in checking, at all positions in the text between 0 and *n*-*m*, whether an occurrence of the pattern starts there or not.
* The brute force algorithm requires no preprocessing phase, and a constant extra space in addition to the pattern and the text
* The expected number of text character comparisons is 2n

**ADVANTAGES:**

* Easy to think of
* Easy to implement
* Usually helps you understand the problem, allowing you to come up with better solutions
* One always exists unless the solution space is infinite

**DISADVANTAGES:**

* Slower than any other types of algorithms

**APPLICATIONS**:

* check the weak passwords used in the system, network or any other applications.

**GREEDY ALGORITHM:**

* Greedy method is used to find restricted most favorable result which may finally land in globally optimized answers.
* But usually greedy algorithms do not gives globally optimized solutions.

**ADVANTAGES:**

* **Analyzing the run time for greedy algorithms will generally be much easier** than for other techniques.

**DISADVANTAGES:**

* The difficult part is that for greedy algorithms**you have to work much harder to understand correctness issues**

**APPLICATIONS:**

* Playing games (eg) chess .

**DIVIDE AND CONQUER ALGORITHM:**

* 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.
* one task breaks another task into smaller, more manageable pieces, and then takes control of those pieces one by one.

**ADVANTAGES:**

* In a perfect world, where the problem is easy to divide, and the sub-problem at some level is easy to solve, divide and conquer can be optimal for a general case solution, like merge sort.
* Parallel availability, divide and conquer by it is very nature lends itself well to parallel processing.

**DISADVANTAGES:**

* Problem decomposition may be very complex and thus not really suitable to divide and conquer.
* Recursive nature of the solution may end up duplicating sub-problems, dynamic solutions may be better in some of these cases, like Fibonacci.
* Recursion into small base cases may lead to huge recursive stacks, and efficiency can be lost by not applying solutions earlier for larger base cases.

**APPLICATIONS:**

* Projects management, development
* Merge sort
* Quick sort

**DYNAMIC PROGRAMING ALGORITHMS:**

* The dynamic programming languages which, at runtime, execute many common programming behaviors that static programming languages perform during compilation.

**ADVANTAGES:**

* Easy to track back and change the solutions dynamically and findout the best among the obtained output, if it flees not up to the mark then again it is possible to change and implement.
* full of expressiveness, save time on writing and compiling code

**DISADVANTAGES:**

* Greater risk of error in run-time.
* slow execution speed in general.

**APPLICATIONS:**

* Decision making  
  Query optimization
* Using Excel

**BACKTRACKING ALGORITHMS**:

* Backtracking is a general algorithm  for finding all solutions to some computational problems.
* A backtracking algorithms  ends when there are no more solutions to the first sub-problem.

**ADVANTAGES:**

* It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
* An algorithm uses a definite procedure.
* It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
* Every step in an algorithm has its own logical sequence so it is easy to debug.
* By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program.

**DISADVANTAGES:**

* Algorithms is Time consuming.
* Difficult to show Branching and Looping in Algorithms.
* Big tasks are difficult to put in Algorithms.

**APPLICATIONS:**

* Eight queens puzzle.
* Constraint satisfaction problem.
* Sudoku solving algorithm.