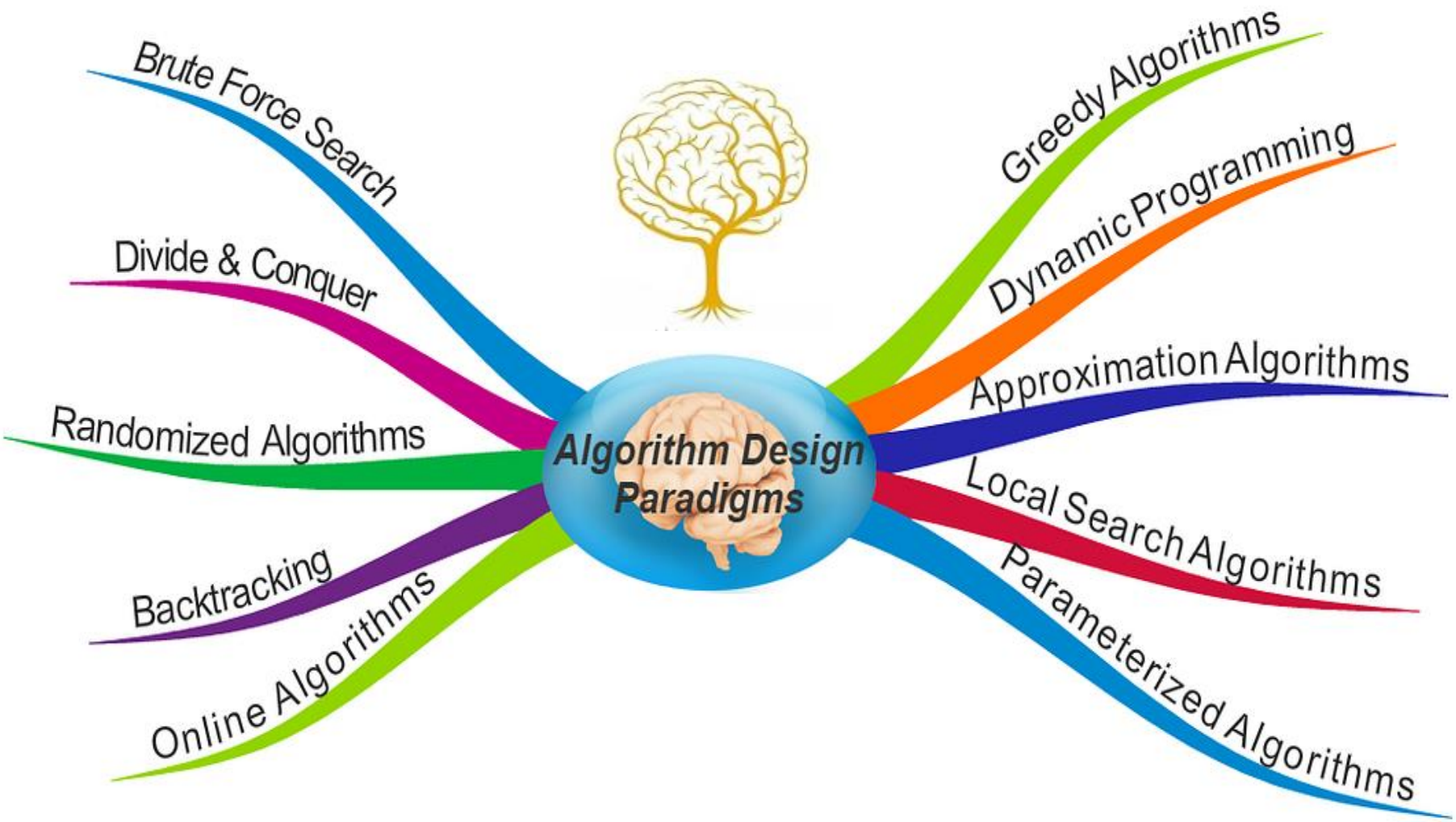

It is a Finite Set of Instructions to solve a Specific Problem

Different Methodology of DAA

- **1. Divide and Conquer Approach:** (Searching, Sorting, Heap Tree)
- **2. Greedy Technique: (**Job sequensing, Knapsack, Optimal Merge Pattern, Halfman's Coding, MST, Dijkstra**)**
- **3. Dynamic Programming:** (APSP, Multistage Graph, TSP, Optimal Merge Pattern, Matrix Chain..etc)
- **4. Branch and Bound/ Graph traversal: (**DFT,BFS, Connected Components**)**
- **5. Backtracking Method: (**N-Queen, Sudoku, Puzzle solving **)**

Fundamental parameters based on which we can analysis the algorithm:

- **Space Complexity:** The space complexity can be understood as the amount of space required by an algorithm to run to completion.

- **Time Complexity:** Time complexity is a function of input size **n** that refers to the amount of time needed by an algorithm to run to completion.

Measures to Analyze an Algorithm

- **Asymptotic analysis:** This sort of analysis measures the overall performance of an algorithm as the enter size methods infinity. It normally includes the usage of mathematical notation to describe the boom fee of the algorithm's strolling time or area usage, including O(n), Ω(n), or Θ(n).

a. Worst-case evaluation: This type of analysis measures the worst-case running time or space utilization of an algorithm, assuming the enter is the maximum toughest viable for the algorithm to deal with.

b. Average-case analysis: This kind of evaluation measures the predicted walking time or area usage of an algorithm, assuming a probabilistic distribution of inputs.

c. Best-case evaluation: This form of analysis measures the nice case running time or area utilization of an algorithm, assuming the input is the easiest possible for the algorithm to address.

- **ALGORITHM Prim(G)**
- **//Prim's algorithm for constructing a minimum spanning tree**
- **//Input: A weighted connected graph G = V,E**
- **//Output: minimum spanning tree T of G**
- **T ← {0} //the set of tree vertices can be initialized with U U ← {1}**
- **for i ← 1 to |V | − 1 do**
- **let (u,v) be the lowest cost edge such that**
- **u€U and v € V-U**
- **T ← T ∪ {(u,v)} // union add edge to spanning tree**
- **U ← U ∪ {v}**
- **Return T**
- **End Prim**