1. Uninformed Search Algorithms

These algorithms do not have additional information about the goal node beyond the problem definition.

- **Linear Search (Sequential Search)**
  - Searches through the list sequentially.
- **Binary Search**
  - Efficient search for sorted arrays, divides the search space by half.
- **Breadth-First Search (BFS)**
  - Explores all neighbors at the present depth prior to moving on to nodes at the next depth level.
- **Depth-First Search (DFS)**
  - Explores as far along each branch as possible before backtracking.
- **Uniform-Cost Search (UCS)**
  - A BFS variation that uses priority queue, searching the least cumulative cost node.
- **Depth-Limited Search**
  - DFS with a predetermined limit to the depth of search.
- **Iterative Deepening Search (IDS)**
  - Combines depth-limited search and BFS to find the shallowest solution.
- **Bidirectional Search**
  - Runs two simultaneous searches: one forward from the start and the other backward from the goal, meeting in the middle.

2. Informed (Heuristic) Search Algorithms

These algorithms use heuristics to make informed decisions about which paths to explore.

- **Best-First Search**
  - Uses a priority queue to explore the most promising nodes first based on a heuristic.
- *A Search**
  - Combines UCS and Best-First Search using a heuristic and path cost function (`f(n) = g(n) + h(n)`).
- **Greedy Best-First Search**
  - Prioritizes nodes with the lowest heuristic value, ignoring the cost to reach them.
- **Beam Search**
  - A variation of Best-First Search that only keeps a limited number of best nodes at each level.
- **Hill Climbing**
  - Iteratively searches for a better neighboring state until no better states are found.
- **Simulated Annealing**
  - Variation of Hill Climbing that probabilistically allows moves to worse states to escape local optima.
- *Iterative Deepening A (IDA*)**
  - Combines A* search and Iterative Deepening Search.
- **Recursive Best-First Search (RBFS)**
  - Optimizes memory usage in Best-First Search by using recursion and maintaining only a limited set of nodes.

3. Local Search Algorithms

These focus on searching for a solution within a localized region of the state space, often for optimization problems.

- **Tabu Search**
  - An advanced form of Hill Climbing with memory to avoid revisiting previously explored states.
- **Genetic Algorithms**
  - Mimics the process of natural evolution by using crossover, mutation, and selection.
- **Random Search**
  - Generates and evaluates random solutions in the search space.
- **Simulated Annealing**
  - Uses randomness to escape local optima and converges to a solution.
- **Gradient Descent**
  - Moves towards the direction of steepest descent to minimize a function.

4. Exponential Search Algorithms

These algorithms are used when searching in infinite or unbounded data sets.

- **Exponential Search**
  - Extends Binary Search by first finding the range where the target element is located.
- **Jump Search**
  - Similar to Binary Search but jumps ahead by a fixed number of steps and then performs a linear search.

5. Interpolation and Adaptive Search Algorithms

These algorithms adjust based on the data being searched.

- **Interpolation Search**
  - Improves Binary Search by estimating the position of the target based on a linear interpolation formula.
- **Fibonacci Search**
  - Uses the Fibonacci sequence to divide the search range, more optimal than Binary Search in certain cases.

6. Hash-Based Search

These algorithms use hashing to achieve constant-time lookups.

- **Hash Search**
  - Searches for a key in a hash table, offering average O(1) time complexity.

7. Tree-Based Search Algorithms

These search algorithms are typically used with tree data structures.

- **Binary Search Tree (BST) Search**
  - Searches for a node in a binary search tree with an average O(log n) time complexity.
- **AVL Tree Search**
  - Searches a self-balancing binary search tree, maintaining O(log n) search time.
- **Red-Black Tree Search**
  - Similar to AVL, but with more lenient balancing criteria, ensuring O(log n) search.
- **B-Tree Search**
  - Generalization of a binary search tree that allows more than two children, used in databases.

8. Probabilistic Search Algorithms

These algorithms involve randomness or probability.

- **Randomized Search**
  - Uses random choices in the search process to explore the search space.
- **Las Vegas Algorithm**
  - Always produces the correct result but varies in runtime due to random choices.
- **Monte Carlo Search**
  - Uses random sampling to solve problems, often in decision-making.

9. Pattern Matching Algorithms

These are used to search for substrings or patterns in strings.

- **Knuth-Morris-Pratt (KMP) Algorithm**
  - Avoids redundant comparisons by preprocessing the pattern.
- **Rabin-Karp Algorithm**
  - Uses hashing to find a substring in a string in an average O(n + m) time complexity.
- **Boyer-Moore Algorithm**
  - Efficiently skips sections of the text using character heuristics and preprocessing.
- **Aho-Corasick Algorithm**
  - A multi-pattern search algorithm that uses a finite state machine for searching multiple patterns simultaneously.

10. Search Algorithms in Graphs

These algorithms are specific to graph traversal.

- **Dijkstra’s Algorithm**
  - Finds the shortest path from a source to all vertices in a weighted graph.
- **Bellman-Ford Algorithm**
  - Similar to Dijkstra but works with graphs that have negative weights.
- **Floyd-Warshall Algorithm**
  - Solves the all-pairs shortest path problem in weighted graphs.
- **Prim's Algorithm**
  - Finds the minimum spanning tree of a graph.
- **Kruskal's Algorithm**
  - Another algorithm to find the minimum spanning tree, often used in sparse graphs.