Eclipse, Kliezl P.
2018-53051
CMSC 123 A.Y. 2019-2020

Paradigm	Description	Operating Principle	Applicable when	E.g., Algorithm
1. Brute Force and Exhaustive Search	The “force” in its name implies the amount of consideration it takes regardless of how large or small it is, as long as the problem is solved. In simple terms, “just do it” best describes this algorithm technique.	Brute force is a straightforward approach in solving a particular problem based on its statement and definition of concepts involved which considers every possible situation and relies on total computing power.	Brute force is applicable to a wide variety of problems, and it can serve an important educational purpose as a standard for comparison in order to find and judge more efficient approaches in solving a problem.	1. Selection Sort - An in-place comparison sorting algorithm wherein the list is divided into two parts – the sorted list on the left side, and the unsorted on the right side 2. Bubble Sort - A sorting algorithm wherein adjacent elements are exchanged when in wrong order 3. Sequential Search - An algorithm which compares successive elements in a given list with a certain search key until it finds a match or the list is exhausted without finding a match 4. Brute-Force String-Matching Algorithm 5. Definition-based algorithm for matrix multiplication

				6. Depth-first Search and Breadth-first Search - Two principle graph-traversal algorithms
2. Decrease-and-Conquer	Decrease-and-conquer is an approach to a given problem by reducing (or decreasing) it and come up with a solution for the smaller instance.	Decrease-and-Conquer technique is based on making use of the relationship (either top-down our bottom-up) between the solution of an instance of a problem and the solution to its smaller instance. It commonly leads to a recursive implementation when using the top-down variation and leads to an incremental approach when using the bottom-up variation. Decrease-and-Conquer technique has 3 major variations: (a) Decrease by a Constant – the size of an instance is reduced to a constant on each iteration, (b) Decrease by a Constant Factor – a problem instance is reduced by the same constant factor, and (c) Variable-size-	Decrease-and-Conquer approach is somewhat like Divide-and-Conquer approach, but instead of generating the problem to two or more subproblems, the problem is reduced to a single problem smaller than the original.	1. Insertion Sort - A direct application of the decrease-(by one)-and-conquer technique to the sorting problem 2. Topological Sorting - Lists vertices of a digraph in an order such that for every edge of the digraph, the vertex it starts at is listed before the vertex it points to 3. Binary Search - A principal example of a decrease-by-a-constant-factor algorithm 4. Interpolation Search

		decrease – the size-reduction pattern varies from one iteration to the other.		
3. Divide-and-Conquer	Divide-and-conquer is named for dividing a given problem to subproblems to obtain its solution.	The Divide-and-Conquer approach follows a general plan: A problem is divided into subproblems of the same type, each are solved recursively, then the solutions to the subproblems are combined to get a solution to the original problem.	Divide-and-Conquer approach is suitable for problems that can be solved by parallel computing – subproblems are solved simultaneously by its own processor.	1. Merge sort - A divide-and-conquer sorting algorithm which divides an input array into two halves, sorts them recursively, and then merges the two 2. Quick Sort - A divide-and-conquer sorting algorithm that works by partitioning its input elements according to their value relative to some preselected element 3. Binary Tree Traversals Preorder; Inorder; and Postorder 4. Strassen's Algorithm - can multiply two n × n matrices with about $n^{2.807}$ multiplications.
4. Transform-and-Conquer	Transform-and-Conquer technique is named for	Transform-and-Conquer is an approach which deals with two-stage procedures:	Transform-and-Conquer approach is best if a problem's instance can be	For Instance Simplification: - List Presorting;

	solving a problem based on the idea of transformation.	Transformation Stage where the instance of a problem is modified to be more amenable to solution, and the Conquering Stage where it is then solved.	transformed to one of three major variants: 1. Instance Simplification (Transform to a simpler instance of the same problem) 2. Representation Change (Transform to another representation) 3. Problem Reduction (Transform to an instance of a different problem with an available algorithm)	- Gaussian Elimination (an algorithm for solving systems of linear equations by transforming it to an equivalent system with an upper-triangular coefficient matrix); and - Rotations in AVL Tree For Representation Change: - Representation of a set by a 2-3 tree; - Heapsort (sorting algorithm based on arranging elements of an array in a heap and then successively removing the largest element from a remaining heap); and - Horner’s rule for polynomial evaluation (an optimal algorithm for polynomial evaluation without coefficient preprocessing, which requires only *n* multiplications and *n* additions to evaluate an *n*-degree polynomial at a given point) For Problem Reduction - Computing the least common multiple;

				- Counting paths in a graph; - Reduction of Optimization Problems; - Linear Programming; and - Reduction to Graph Problems
5. Dynamic Programming	"Dynamic" in its name is referred to the time-varying aspect of the problem, while "Programming" in its name is referred to "planning" as it was invented as a general plan for optimizing multistage decision processes.	Dynamic Programming is a technique for solving problems with overlapping subproblems. Instead of repeatedly solving these subproblems, the technique suggests performing the solution to a smaller problem once, then recording the result in a table from which a solution to the original problem can be obtained.	Dynamic Programming is best used when problems satisfy the *principle of optimality* (the problem is divided to subproblems, and their results can be reused).	1. Solving the change-making problem by dynamic programming 2. Solving the knapsack problem by dynamic programming 3. Construction of Optimal Binary Search Tree 4. Floyd-Warshall's algorithm - Warshall's Algorithm finds the transitive closure, and Floyd's Algorithm solves the all-pairs shortest-path problem.
6. Greedy Technique	On each step, the algorithm greedily chooses the best alternative available, hoping that a sequence of locally optimal solution will lead	The goal of the Greedy Technique is to construct a solution through a sequence of steps, which creates a partially constructed solution, until a completed solution is reached. On each step, a choice needs to satisfy the following	Using Greedy Technique is best when a problem satisfies two properties: (a) The Greedy-choice Property, in which the global optimum can be reached by selecting a local optimum, and	1. Prim's Algorithm - A greedy algorithm for constructing a minimum spanning tree of a weighted connected graph which works by attaching to a previously constructed subtree a vertex closest to the vertices already in the tree

	to a globally optimal solution.	criteria: *feasible*, *locally optimal*, and *irrevocable*.	(b) The Optimal Substructure, whose optimal solution contains optimal solutions to its subproblems.	2. Kruskal's Algorithm - A greedy algorithm for the minimum spanning tree problem, which constructs a minimum spanning tree by selecting edges in nondecreasing order of their weights provided that the inclusion does not create a cycle 3. Dijkstra's algorithm - An algorithm that solves the single-source shortest-path problem of finding shortest paths from a given vertex (the source) to all the other vertices of a weighted graph or digraph 4. Huffman codes - An optimal prefix-free variable-length encoding scheme that assigns bit strings to symbols based on their frequencies in a given text
7. Iterative Improvement	Iterative-Improvement Technique builds an optimal solution	Iterative Improvement technique starts with a feasible solution,	Iterative-Improvement Technique is frequently used in numerical problems such as finding	1. Simplex Method - A method for solving a general linear programming problem

	by generating a sequence of feasible solutions (solutions that satisfy the constraints of the problem) with improving values of the problem's objective function.	proceeds to improve it by repeating applications of some simple step which involves a small change, then stops when no such change improves the value of the objective function.	the maximum number of a function. Example problems that can be solved with this are: linear programming, finding the maximum flow in a graph/network, and matching the maximum possible number of vertices in a graph.	2. Ford-Fulkerson method - A template that solves the maximum flow problem 3. Shortest augmenting-path method - Implements the idea behind Ford-Fulkerson method by labeling vertices via breadth-first search 4. Gale-Shapley Algorithm - An algorithm which can find a stable matching
8. Backtracking	Backtracking technique comes to a point when it performs candidates of solutions and "backtracks" as soon as it determines the candidate performs an infeasible solution.	The idea behind the Backtracking technique is to construct solutions a component at a time and evaluate the candidates. A partially constructed solution is further developed if there are no concerns regarding the given problem's constraints by taking the first legitimate option for the next component. In case there is no option for the next component, then it backtracks to replace the last component of the partially constructed solution with the	Backtracking technique is applicable to solving constraint satisfaction problems or puzzles such as: - Eight queens puzzle - Knight's tour - Logic programming languages - Crossword puzzles, and - Sudoku. Backtracking technique is not constrained by the	1. Implementation of Sudoku 2. Solving the n-Queens problem 3. Finding a Hamiltonian Circuit in a graph 4. Solving the Subset-Sum Problem

		next option. Backtracking technique is convenient to be implemented as a *state-space tree* (a tree of choices being made) in a manner of *Depth-first search*. A node is considered *promising* if it corresponds to a partially constructed solution and may lead to a complete solution. The leaves of this tree may be complete solutions or nonpromising dead ends.	demands applicable for the Branch-and-bound technique, but more often than not, it applies to non-optimization problems.	
9. Branch-and-bound	Branch-and-bound technique, similar to Backtracking, cuts off a branch of the problem's state-space tree when it leads to an infeasible solution. A bound value is included in this tree and serves as the basis for a node to lead an optimal solution or not.	The Branch-and-bound technique enhances the idea of generating a state-space tree with the idea of estimating the best value obtainable from a current node of the decision tree. The principal idea behind it is no solution obtained from it can yield a better solution than the one already available.	Branch-and-bound technique is only applicable to optimization problems since it is based on computing a bound on possible values of the problem's objective function.	1. Solving the Assignment Problem by Branch-and-bound technique 2. Solving the Knapsack Problem 3. Solving the Traveling Salesman Problem