Assignment 1:

1. Dynamic Programming vs Greedy algorithm

Feature/Aspect	Dynamic Programming (DP)	Greedy Algorithm
Concept	Breaks problem into overlapping subproblems	Makes the locally optimal choice at each step
Approach	Bottom-Up (Tabulation) or Top-Down (Memoization)	Iterative, making a series of choices
Subproblem Solutions	Reuses solutions to subproblems	Does not reuse subproblems, makes a single pass
Time Complexity	Usually polynomial (depends on the problem)	Usually more efficient, often linear or log-linear
Space Complexity	Can be high due to storage of subproblem solutions	Generally lower, as it doesn't store solutions to subproblems
Optimal Substructure	**Yes, solution constructed from optimal solutions of subproblems**	**Yes, solution is built using locally optimal choices**

2. Greedy Method and divide and conquer algorithm

Feature/Aspect	Greedy Method	Divide and Conquer
Concept	Makes the locally optimal choice at each step	Divides the problem into smaller subproblems, solves them independently, and combines their solutions
Approach	Iterative, making a series of choices	Recursive, solving subproblems and merging their results
Subproblem Solutions	Does not typically reuse subproblems	Solves subproblems independently and combines their results
Optimal Substructure	Solution is built using locally optimal choices	Solution is built by combining optimal solutions of subproblems
Overlapping Subproblems	No, typically does not involve overlapping subproblems	No, subproblems are solved independently
Solution Guarantee	May or may not find the globally optimal solution	Finds the globally optimal solution if the problem can be decomposed correctly
Examples	Prim's and Kruskal's algorithms for Minimum Spanning Tree	Merge Sort, Quick Sort
	Dijkstra's algorithm for shortest paths (with non-negative weights)	Binary Search
	Huffman Coding	Fast Fourier Transform (FFT)
Time Complexity	Usually more efficient, often linear or log-linear	Varies, often \(O(n \log n)\) for sorts, can be worse in other cases
Space Complexity	Generally lower, as it doesn't store solutions to subproblems	Can be high due to recursive calls and storage of subproblems
When to Use	When a problem can be solved by making a series of local, greedy choices	When a problem can be broken down into smaller, independent subproblems
Recursive Nature	Typically non-recursive	Recursive
Combining Solutions	Not required, as each choice is final	Combines solutions of subproblems

3. Divide and Conquer vs Dynamic Programming

Feature/Aspect	Divide and Conquer	Dynamic Programming
Concept	Divides the problem into smaller subproblems, solves them independently, and combines their solutions	Solves complex problems by breaking them into simpler overlapping subproblems and solving each subproblem only once
Approach	Recursive, solving subproblems and merging their results	Bottom-Up (Tabulation) or Top-Down (Memoization)
Subproblem Solutions	Solves subproblems independently and combines their results	Reuses solutions to subproblems
Optimal Substructure	Solution is built by combining optimal solutions of subproblems	Solution is constructed from optimal solutions of subproblems
Overlapping Subproblems	No, subproblems are solved independently	Yes, subproblems are solved multiple times
Solution Guarantee	Finds the globally optimal solution if the problem can be decomposed correctly	Always finds the globally optimal solution if the problem has optimal substructure and overlapping subproblems
Examples	Merge Sort, Quick Sort	Fibonacci sequence, Knapsack problem (0/1 and fractional)
	Binary Search, Fast Fourier Transform (FFT)	Longest Common Subsequence (LCS)
Time Complexity	Varies, often \(O(n \log n)\) for sorts, can be worse in other cases	Usually polynomial (depends on the problem)
Space Complexity	Can be high due to recursive calls and storage of subproblems	Can be high due to storage of subproblem solutions
When to Use	When a problem can be broken down into smaller, independent subproblems	When the problem has overlapping subproblems and optimal substructure
Recursive Nature	Recursive	Can be recursive (Memoization) or iterative (Tabulation)
Combining Solutions	Combines solutions of subproblems	Combines solutions of subproblems

4. Dynamic Programming vs Backtracking

Feature/Aspect	Dynamic Programming	Backtracking
Concept	Solves complex problems by breaking them into simpler overlapping subproblems and solving each subproblem only once	Explores all possible solutions to a problem, backtracks when a solution path fails to satisfy constraints
Approach	Bottom-Up (Tabulation) or Top-Down (Memoization)	Recursive, explores all possible solution paths
Subproblem Solutions	Reuses solutions to subproblems	Explores each solution path individually, does not reuse subproblems
Optimal Substructure	Solution is constructed from optimal solutions of subproblems	Does not necessarily exhibit optimal substructure
Overlapping Subproblems	Yes, subproblems are solved multiple times	No, explores each solution path separately
Solution Guarantee	Always finds the globally optimal solution if the problem has optimal substructure and overlapping subproblems	Finds a solution if one exists, but not necessarily the optimal one
Examples	Fibonacci sequence, Knapsack problem (0/1 and fractional) Longest Common Subsequence (LCS)	N-Queens problem, Sudoku solver Combination and permutation problems
Time Complexity	Usually polynomial (depends on the problem)	Can be exponential in the worst case
Space Complexity	Can be high due to storage of subproblem solutions	Depends on the depth of the recursion tree, can also be high
When to Use	When the problem has overlapping subproblems and optimal substructure	When a problem requires exploring all possible solutions and constraints can be applied to prune paths
Recursive Nature	Can be recursive (Memoization) or iterative (Tabulation)	Recursive
Combining Solutions	Combines solutions of subproblems	Does not combine solutions, finds a solution through exploration