**Name: G.Madhulatha Reg.No:192311295 Course:DAA Course Code:CSA0614**

**1. Analyse Time Complexity of a Brute-Force Approach**

The brute-force approach would involve evaluating all possible pricing combinations to maximize profit. Given a large product catalog and multiple prices per product, this quickly becomes computationally infeasible due to exponential time complexity. If there are n*n* products and m*m* possible price points per product, the time complexity is O(mn)*O*(*mn*), which grows extremely fast as n*n* increases. This analysis will help justify the need for more efficient strategies like dynamic programming and approximation algorithms.

**2. Prove Correctness of Dynamic Programming and Greedy Strategies**

* **Dynamic Programming (DP)**: A DP approach can break down the pricing problem into subproblems that optimize pricing for each product based on demand, supply, and competitor prices. To prove correctness:
  + Show that the subproblems are optimal independently and that combining solutions maintains optimality.
  + Use induction to confirm that each decision optimally maximizes profit for any set of constraints.
* **Greedy Strategy**: If appropriate (e.g., for specific subsets of products where local decisions lead to a global optimum), a greedy strategy could yield efficient pricing updates. Proving correctness would involve demonstrating that making the "locally best" price decision for each product guarantees the global profit maximization.

**3. Implementing Algorithms**

* **Brute-Force**: Implement to benchmark performance, understanding limits on execution time and scalability.
* **Dynamic Programming (DP)**: This can focus on iterative, state-based profit maximization for each product and its constraints, producing optimal or near-optimal pricing within polynomial time for certain cases.
* **Approximation Algorithms**: Given real-time constraints, use approximation techniques (e.g., heuristics, machine learning-based predictions) to balance accuracy and time complexity. Approximation approaches can achieve acceptable pricing accuracy with O(nlog⁡n)*O*(*n*log*n*) or lower time complexity in certain scenarios.

**4. Apply Backtracking for Specific Scenario Testing**

Backtracking will allow exploration of alternative price combinations in niche scenarios, such as high-demand fluctuations or product shortages. This involves recursively testing different pricing paths and can reveal local optima that approximate a solution close to the global optimum.

**5. Compare Polynomial and Non-Polynomial Algorithms for Scalability**

Scalability testing involves comparing algorithms in terms of:

* **Execution Time**: Measure how each algorithm scales with catalog size.
* **Memory Usage**: Assess the memory footprint of dynamic programming and approximation algorithms.
* **Accuracy**: Determine the trade-off between polynomial-time approximations versus the exact solutions from non-polynomial approaches (e.g., brute-force).

**Deliverables**

1. **Code Implementation**:
   * Brute-force algorithm for benchmark purposes.
   * Dynamic programming and greedy algorithms for optimal or near-optimal pricing.
   * Approximation-based algorithms for efficient, real-time adjustments.
2. **Report on Trade-offs**:
   * Comparison of time complexity, accuracy, and performance.
   * Pros and cons of each method for large product sets.
   * Considerations for balancing profit maximization with real-time feasibility.
3. **Visualizations**:
   * Graphs and charts demonstrating revenue impact for each algorithm across varying demand levels.
   * Analysis of how changes in supply, demand, and competitor pricing affect profitability across different approaches.