Algorithm Basics - Recap

Correction

1. **Planning an efficient route for a road trip**

   Dynamic Programming - It is suitable here due to the presence of overlapping subproblems (calculating the shortest route from one point to another) and the need for an optimal solution. It systematically breaks down the problem into smaller parts, remembers past results (memoization), and ensures the most efficient route is found.

   Randomized - A Monte Carlo method could be used to handle the route optimization by randomly sampling different routes and estimating the best one. This approach is beneficial when the problem space is too large for exhaustive search and where probabilistic methods can provide near-optimal solutions in a reasonable time frame.

2. **Organizing a bookshelf efficiently by genre, author etc.**

   Divide and Conquer - It is ideal for sorting as it breaks the problem into smaller, more manageable parts (like sorting smaller sections of books) and then combines them. This approach, seen in sorting algorithms like Merge Sort, efficiently manages the complexity of sorting large numbers of books.

   Greedy Algorithm - A Greedy algorithm could provide a quick, although not necessarily optimal, way to organize books. It might involve picking books and placing them in the 'best' location available at that moment, like grouping by the most obvious category first. This approach is simple and fast but may not yield the most organized shelf.

3. **Designing a school timetable that schedules classes and teachers**

   Greedy Algorithm - It is appropriate for timetabling because it makes local optimal choices at each step (e.g., assigning the most demanding classes to the most available time slots first) with the aim of finding a global optimum. It's efficient in scenarios where the best immediate decision is also a good approximation for the overall best decision.

   Randomized - A Las Vegas-type algorithm randomly tries different combinations of classes and teachers until a feasible timetable is found. This approach can be useful when there are too many constraints for a greedy algorithm to handle effectively and when an element of randomness can help escape local optima.

4. **Budgeting for a family vacation by allocating funds to activities**

   Dynamic Programming - Budgeting involves optimizing resource allocation with overlapping subproblems (e.g., allocating funds across days or activities where decisions on one day affect the next). Dynamic programming would efficiently find the optimal allocation by breaking the problem into smaller, manageable subproblems and solving each just once.

   Greedy Algorithm - For simpler budgeting scenarios, a Greedy approach can work well, where decisions are made based on the most attractive activity or deal at the moment. It's quick and easy to implement, although it might not always provide the most cost-effective distribution of funds.

5. **Optimizing space utilization in a truck**

   Greedy Algorithm - Greedy approach makes sense for a quick, locally optimal solution (e.g., packing the largest or heaviest items first).

   Brute Force - For an optimal arrangement that maximizes space usage, a Brute Force method might be necessary, especially for complex packing with many differently shaped items, albeit being time-consuming.

   Divide and Conquer - It can be applied by dividing the truck space into sections and solving the packing problem for each section independently. This can be more methodical than a greedy approach and can handle complex packing requirements better than brute force, though it may not always be as optimal.

6. **Finding the best deals in a sale by comparing discounts on various products**

   Greedy Algorithm - It is suitable for making quick decisions that seem best at the moment (e.g., picking items with the highest discount first). It's efficient for shoppers making immediate decisions based on visible benefits without needing to consider every possible combination of purchases.

   Brute Force - For smaller datasets or simpler sale scenarios, a Brute Force approach could systematically evaluate every possible combination of purchases to find the absolute best deal. This method guarantees finding the best deal but is time-consuming and impractical for large datasets.