Problem Description: Minimum Coin Change Problem

The minimum coin change problem is a classic problem in computer science and optimization. Given a set of coin denominations and a target amount, the task is to find the minimum number of coins needed to make change for that amount. The objective is to minimize the total number of coins used.

For example, consider the coin denominations {1, 2, 3} and a target amount of 7. The optimal solution would be to use two coins of denomination 3 and one coin of denomination 1, totaling three coins.

Greedy Solution

The greedy solution for the minimum coin change problem involves iteratively selecting the largest possible coin at each step until the target is reached. The algorithm sorts the coin denominations in descending order and then iterates through them, subtracting the largest possible denomination from the target until the target becomes zero.

Asymptotic Upper Bound

- **Time Complexity**: **O(n log n)**, where n is the number of coin denominations. The dominant factor is the sorting of the coins, it is due to sorting.

- **Space Complexity**: **O(1),** as the algorithm uses a constant amount of space.

Advantages: Simple and intuitive. Efficient for certain scenarios where the greedy-choice property holds.

Disadvantages: Not always optimal; may fail for some cases. Does not guarantee the global optimum.

Dynamic Programming Solution

The dynamic programming solution for the minimum coin change problem uses a table (`dp`) to store the minimum number of coins needed for each target amount. It iteratively fills the table based on the optimal substructure of the problem.

For the first row (i = 0), initialize dp[0][j] based on the coin denomination coins[0]. If j is divisible by coins[0], set dp[0][j] to the number of coins needed, otherwise, set it to a large value (INT_MAX) indicating it's not possible. For the remaining rows (i > 0), iterate through each target amount (j) and consider two options:

Not taking the current coin (notTake): The minimum number of coins needed for the current target without considering the current coin. This value is obtained from the row above (dp[ind - 1][target]).

Taking the current coin (take): The minimum number of coins needed for the reduced target (target - coins[ind]) plus one (the current coin).

Update dp[ind][target] as the minimum of these two options. The minimum number of coins needed to make change for the target amount T is stored in dp[n - 1][T]. If this value is equal to INT_MAX, it means it's not possible to make change, and -1 is returned. Otherwise, the result is returned.

Asymptotic Upper Bound

- **Time Complexity**:**O(nT),** where n is the number of coin denominations and T is the target amount. The algorithm fills a table of size n x T.

- **Space Complexity**: **O(nT),** as the algorithm uses a 2D table of size n x T.

Advantages: Guarantees optimal solution. Handles a wide range of scenarios.

Disadvantages: More complex and may have higher constant factors. Requires more space.

In summary, the choice between the two solutions depends on the problem characteristics. If the problem satisfies the greedy-choice property, the greedy solution might be efficient. Otherwise, the dynamic programming solution is a more general approach that guarantees the optimal solution. The time and space complexities should be considered based on the problem size and constraints.