Experiment: Greedy Coin Change Implementation:

1. Objective

To implement the Coin Change problem using a greedy approach in C, evaluate its performance for standard US coin denominations, and analyze its time and space complexity.

2. Algorithm

The Coin Change problem involves finding the minimum number of coins to make a given amount using a set of denominations. The greedy approach:

- Uses a fixed set of denominations (e.g., US coins: 25, 10, 5, 1).
- Iterates through denominations in descending order.
- For each denomination, uses as many coins as possible without exceeding the remaining amount.
- Updates the remaining amount and counts total coins used.

3. Theoretical Solution

- **Problem**: Given an amount and coin denominations, determine the minimum number of coins needed.
- **Greedy Approach**:
    - Start with the largest denomination.

    - Use as many coins of that denomination as possible.
    - Reduce the amount and proceed to the next denomination.
    - Repeat until the amount is zero.
- **Correctness**: Optimal for US coin denominations (25, 10, 5, 1) due to their structure, but not guaranteed for arbitrary denominations.
- **Time Complexity**: O(n), where n is the number of denominations (constant for fixed US coins, O(1) in practice).
- **Space Complexity**: O(1), as only a fixed array and a few variables are used.
-

4. Practical Work

a. Pseudocode

START

DEFINE coins as [25, 10, 5, 1]

DEFINE n as number of coin types (length of coins array)

SET totalCount = 0

DISPLAY "Enter the amount: "

READ amount

DISPLAY "Change for", amount

FOR i FROM 0 TO n - 1 DO

IF amount >= coins[i] THEN

SET count = amount DIV coins[i]

SET amount = amount - (count * coins[i])

INCREMENT totalCount by count

DISPLAY count, "coin(s) of", coins[i]

ENDIF

END FOR

DISPLAY "Total count coins = ", totalCount

END

b. Source Code

```
#include <stdio.h>



void greedycoinchange(int amount) {
    int coins[] = {25, 10, 5, 1};
    int n = sizeof(coins) / sizeof(coins[0]);
      int c=0;
    printf("Change for %d:\n", amount);

    for (int i = 0; i < n; i++) {
        if (amount >= coins[i]) {
            int count = amount / coins[i];

            amount -= count * coins[i];
            c+=count;
```

```
            printf("%d coin(s) of %d\n", count, coins[i]);
        }

    }
    printf(" total count coins = %d",c);
}


int main() {
    int amount;
    printf("enter the amount: ");
    scanf("%d", &amount);

    greedycoinchange(amount);

    return 0;
}
```

Output:

enter the amount: 10

Change for 10:

1 coin(s) of 10

total count coins = 1

5. Analysis Table

Algorithm	Best Case	Worst Case	Avg Case	Space
Greedy	O(1)	O(1)	O(1)	O(1)

- **Note**: For fixed denominations (n = 4), the time complexity is effectively O(1) as the loop iterates a constant number of times. Space is O(1) due to fixed-size array and variables.

6. Observations

- **Test Case (Amount: 30)**:
  - Output: 1 coin(s) of 25, 1 coin(s) of 5, total count coins = 2
  - Correct for US denominations, matches optimal solution.
- **Test Case (Amount: 67)**:
  - Output: 2 coin(s) of 25, 1 coin(s) of 10, 1 coin(s) of 5, 2 coin(s) of 1, total count coins = 6
  - Correct and optimal.
- **Test Case (Amount: 0)**:
  - Output: total count coins = 0
  - Handles edge case correctly.
- The algorithm consistently produces optimal results for US coin denominations due to their greedy-compatible structure.

7. Challenges

- **Fixed Denominations**: The code is hard-coded for US coins (25, 10, 5, 1), limiting flexibility for other denomination sets.
- **No Error Handling**: Invalid inputs (e.g., negative amounts) are not handled, which could cause undefined behavior.
- **Non-Optimality for Other Systems**: The greedy approach fails for non-standard denominations (e.g., [1, 3, 4] for amount 6), but this is not tested in the provided code.

8. Conclusion

The greedy coin change algorithm implemented in C is highly efficient for US coin denominations, with O(1) time and space complexity due to the fixed number of denominations. It correctly computes the minimum number of coins for standard cases and is simple to implement. However, its applicability is limited to greedy-optimal coin systems, and the code lacks flexibility for dynamic denominations or robust input validation. For general cases, a dynamic programming approach would be necessary to ensure optimality.