Coin Change Problem

- Finding the number of ways of making changes for a particular amount of cents, $n$, using a given set of denominations $C=\{c_1\ldots c_d\}$ (e.g., the US coin system: $\{1, 5, 10, 25, 50, 100\}$)
  - An example: $n = 4, C = \{1,2,3\}$, solutions: $\{1,1,1,1\}$, $\{1,1,2\}$, $\{2,2\}$, $\{1,3\}$.

- Minimizing the number of coins returned for a particular quantity of change (available coins $\{1, 5, 10, 25\}$)
  - 30 Cents (solution: $25 + 2$, two coins)
  - 67 Cents ?

- 17 cents given denominations $= \{1, 2, 3, 4\}$?
Sudoku Puzzle
Find the Fewest Coins: Casher’s algorithm

- Given 30 cents, and coins {1, 5, 10, 25}
- Here is what a cashier will do: always go with coins of highest value first
  - Choose the coin with highest value 25
    - 1 quarter
  - Now we have 5 cents left
    - 1 nickel

The solution is: 2 (one quarter + one nickel)
Greedy Algorithm Does not Always Give Optimal Solution to Coin Change Problem

- Coins = \{1, 3, 4, 5\}
- 7 cents = ?

- Greedy solution:
  - 3 coins: one 5 + two 1

- Optimal solution:
  - 2 coins: one 3 + one 4
Find the Fewest Coins: Divide and Conquer

- 30 cents, given coins \{1, 5, 10, 25, 50\}, we need to calculate MinChange(30)

- Choose the smallest of the following:
  - 1 + MinChange(29) #give a penny
  - 1 + MinChange(25) #give a nickel
  - 1 + MinChange(10) #give a dime
  - 1 + MinChange(5) #give a quarter

- Do not know MinChange(29), MinChange(25), MinChange(10), MinChange(5)?
Coin Change Problem: A Recursive Algorithm

1. **MinChange**($M$)
2. if $M = 0$
3. return 0
4. $v \leftarrow \infty$
5. for $c$ in denominations $\leq M$
6. $v \leftarrow \min \{ \text{MinChange}(M-c) + 1, v \}$
7. return $v$
Recursive Algorithm Is Not Efficient

- It recalculates the optimal coin combination for a given amount of money repeatedly.
How to avoid computing the same function multiple times

- We’re re-computing values in our algorithm more than once
- Save results of each computation for 0 to $M$
- This way, we can do a reference call to find an already computed value, instead of re-computing each time
- Running time $M \times d$, where $M$ is the value of money and $d$ is the number of denominations
1. MinChange($M$)
2. if minChange[M] not empty
3. return minChange[M]
4. if $M = 0$
5. return 0
6. for $c$ in denominations $\leq M$
7. \hspace{1em} $v \leftarrow \min \{\text{MinChange}(M-c) + 1, v\}$
8. minChange[M] = $v$
9. return $v$
The Change Problem: Dynamic Programming

1. MinChangeDP($M$)
2. $minCoin[0] \leftarrow 0$
3. for $m \leftarrow 1$ to $M$
4. $minCoin[m] \leftarrow \text{infinity}$
5. for $c$ in denominations $\leq M$
6. if $minCoin[m-c] + 1 < minCoin[m]$
7. $minCoin[m] \leftarrow minCoin[m-c] + 1$
8. return $minCoin[\text{M}]$
Proof:
Let N be the amount to be paid. Let the optimal solution be
\[ P = A \times 10 + B \times 5 + C. \]
Clearly \( B \leq 1 \) (otherwise we can decrease \( B \) by 2 and increase \( A \) by 1, improving the solution). Similarly, \( C \leq 4 \).

Let the solution given by GreedyCoinChange be
\[ P = a \times 10 + b \times 5 + c. \]
Clearly \( b \leq 1 \) (otherwise the algorithm would output 10 instead of 5). Similarly \( c \leq 4 \).

From \( 0 \leq C \leq 4 \) and \( P = (2A + B) \times 5 + C \) we have \( C = P \mod 5 \).
Similarly \( c = P \mod 5 \), and hence \( c = C \). Let \( Q = (P - C)/5 \).
From \( 0 \leq B \leq 1 \) and \( Q = 2A + B \) we have \( B = Q \mod 2 \).
Similarly \( b = Q \mod 2 \), and hence \( b = B \).
Thus \( a = A \), \( b = B \), \( c = C \), i.e., the solution given by GreedyCoinChange is optimal.