Making Change
Coin Changing

**Goal.** Given currency integer denominations: \{100, 25, 10, 5, 1\} devise a method to pay integer amount to customer using the fewest number of coins.

**Example:** 34¢.

**Cashier's algorithm.** At each iteration, add coin of the largest value that does not take us past the amount to be paid.

**Example:** $2.89 = 289¢.
Coin-Changing: Greedy Algorithm

**Cashier's algorithm.** Use the maximal number of the largest denomination

\[ x \text{ - amount to be changed} \]

Sort coins denominations by value: \( c_1 < c_2 < \ldots < c_n \).

\( \text{S} \leftarrow \emptyset \) \hspace{1cm} \text{coins selected}

while \((x > 0)\) {

let \( k \) be largest integer such that \( c_k \leq x \)

if \((k = 0)\)  

\text{return} \ "no solution found"

\( x \leftarrow x - c_k \)

\( \text{S} \leftarrow \text{S} \cup \{k\} \)

}

\text{return} \ \text{S}

Does this algorithm always work?
Coin-Changing: Greedy doesn't always work

Greedy algorithm works for US coins
Proof: number theory

Greedy fails changing 30 optimally with coin set {25, 10, 1}

Greedy fails changing 30 at all with coin set {25, 10}
Different problem: number of ways to pay

Given a coin set $c = \{c_0, c_1, ..., c_{d-1}\}$ and an amount $M$, how many different ways can $M$ be paid?

Recursive solution: is this a take / don’t take type of problem?

- e.g., for eg 56 cents I can use 0, 1, or 2 quarters

One possible (not the only) solution

Base:
- if $d == 0$, how many ways? (is there always a way ?)

Step:
- if $d > 0$
  - at least how many $c_d$ coins can be used
  - and which problem then remains to be solved?
  - ...
  - at most how many $c_d$ coins can be used
  - and which problem then remains to be solved?

Now turn Recursive into Dynamic Programming
Making Change Recursive

- **d=3**: Quarters
- **d=2**: Dimes
- **d=1**: Nickles
- **d=0**: Cents
Go through the state space bottom-up: \( i=1 \) to \( n \)
- select coin type \( i \),
  - first 1 coin type, then 2, ..., all coin types
  - what does the first column look like?
- use solutions of smaller sub-problems to efficiently compute solutions of larger ones
  - in sss / knapsack there are 2 sub-problems
  - in coins there are how many?

\[ 1 \quad 2 \quad 3 \quad ... \quad i-1 \quad i \quad \text{coins considered} \]