Making Change
Making Change

Goal. Given currency coin denominations, e.g., \{100, 25, 10, 5, 1\} devise a method to pay an integer amount using the fewest 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:
$2.89 = 289¢.$
Cashier's algorithm. Use the maximal number of the largest denomination coin

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

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

\[ S \leftarrow \text{empty} \]

while \( x > 0 \) {
    let \( k \) be largest integer such that \( c_k \leq x \)
    if \( k == 0 \)  \# all \( c_k > x \)
        return "no solution found"
    \( x \leftarrow x - c_k \)
    append(\( S,k \))
}

return \( S \)

Does this Greedy algorithm always work?
Greedy doesn't always work

1. Greedy fails changing \textbf{30 optimally} with coin set \{25, 10, 1\} as it produces \([25,1,1,1,1,1]\) instead of \([10,10,10]\)

2. Greedy fails changing \textbf{30 at all} with coin set \{25, 10\} even though there is a solution: \([10,10,10]\)

3. But Greedy algorithm works for US coin set
Proof: number theory (canonical coin systems)
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?

\[ \text{e.g., for eg 56 cents I can use 0, 1, or 2 quarters} \]

One possible recursive 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?

Can you think of a different algorithm / recurrence?
Making Change Recursive

d=3: Quarters (25 c)
Making Change Recursive

d=3: Quarters (25 c)
d=2: Dimes (10 c)
Making Change Recursive

- **d=3**: Quarters (25 c)
- **d=2**: Dimes (10 c)
- **d=1**: Nickles (5 c)
Making Change Recursive

- **d=3**: Quarters (25 c)
- **d=2**: Dimes (10 c)
- **d=1**: Nickles (5 c)
- **d=0**: Cents (1 c)
**Making Change Dynamic Programming**

**Go through the state space bottom-up: i=0 to n-1**

- select coin type
  - 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

Compare it to SSS / knapsack dynamic programming

In knapsack there are 2 sub-problems, in coins there are how many?

In knapsack you take the max, in coins you take?