Algorithms for Data Science
Lab 5
Stack and Queue ADTs

You are to create two Python classes, MyStack and MyQueue. These should provide a stack and queue implementation, respectively. The MyStack should use a standard Python list as its underlying data structure (make sure it is efficient). The MyQueue should use a Python deque as its underlying data structure (import deque from collections).

Each class should provide methods to create it, add/remove from it, peek at the top/front element without removing it, and check to see if it is empty. The testing code below shows you exactly what your methods should be called and how they should work.

```
# Testing code for stack
s = MyStack(int)
print(s.empty())
s.push(5)
s.push(8)
print(s.pop())
s.push(3)
print(s.empty())
print(s.top())
print(s.pop())
print(s.pop())
print(s.pop()) # should generate an error

# Testing code for Queue
q = MyQueue(int)
print(q.empty())
q.enqueue(5)
q.enqueue(8)
print(q.dequeue())
q.enqueue(3)
print(q.empty())
print(q.front())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue()) # should generate an error
```

```
Enter desired amount of change: 29
DAC:
optimal: 3  in time:  13.3 ms
```

```
DP:
5
12
12
optimal: 3  in time:  0.1 ms
```

```
# Below are two algorithms (DAC and DP) to compute the
# minimum number of coins required to produce A cents worth of change
# The DP version also prints out the coins needed to produce this min

from time import time

# Algorithm 1: Divide-and-Conquer
def DACcoins(coins, amount):
    if amount == 0: # The base case
        return 0
    else:           # The recursive case
        minCoins = float("inf")
        for currentCoin in coins:  # Check all coins
            # If we can give change
            if (amount - currentCoin) >= 0:
                # Calculate the optimal for currentCoin
                currentMin = DACcoins(coins, amount-currentCoin) + 1
                # Keep the best
                minCoins = min(minCoins, currentMin)
        return minCoins

# Algorithm 2: Dynamic Programming with Traceback
def DPcoins(coins, amount):
    # Create the initial tables

    # Fill in the base case(s)

    # Fill in the rest of the table

    # Perform the traceback to print result

    return -1  # return optimal number of coins

C = [1,5,10,12,25] # coin denominations (must include a penny)

A = int(input('Enter desired amount of change: '))
assert A>=0

print("DAC:")
t1 = time()
numCoins = DACcoins(C,A)
t2 = time()
```

```
print("optimal:",numCoins," in time: ",round((t2-t1)*1000,1),"ms")
print()
print("DP:")
t1 = time()
numCoins = DPcoins(C,A)
t2 = time()
print("optimal:",numCoins," in time: ",round((t2-t1)*1000,1),"ms")
```
