```
from time import perf_counter
global storage
storage = {}

#Question 1-3 (uncomment the line below to answer functions from question 1 to 3)
#num = int(input(("Enter a positive integer: ")))

#Question 4 (uncomment the line below to answer functions from question 4)
limit = int(input(("Enter an upper limit: ")))
global tempLimit
tempLimit = limit
#-----------------------------------------------------------
def CollatzConj(n):
    if n == 1:
        print(n)
        return 1
    elif n%2 == 0:
        print(n)
        return CollatzConj(int(n/2))
    else:
        print(n)
        return CollatzConj(n*3 + 1)

#-----------------------------------------------------------
def CollatzConjCount(n):
    if n == 1:
        return 0
    elif n%2 == 0:
        n = int(n/2)
    else:
        n = n*3 + 1
    return 1 + CollatzConjCount(n)
#-----------------------------------------------------------
def memoizer(func, storage):
    def memoized(n):
        if n in storage:
            return storage[n]

        #if it doesn't exist in the storage dictionary add it for future use
        result = func(n)
        storage[n] = result
        return storage[n]
    return memoized(tempLimit)
    #tempLimit can also change to num depending on which question you're on -
simply replace tempLimit with num
#-----------------------------------------------------------
```

```python
def max_collatz_steps(limit):
    max_steps = 0
    for i in range(1, limit + 1):
        global tempLimit
        tempLimit = i
        steps = memoizer(CollatzConjCount, storage)
        if steps > max_steps:
            max_steps = steps
    return max_steps

def max_collatz_num(limit):
    max_steps = 0
    number_with_max_steps = 1
    for i in range(1, limit + 1):
        global tempLimit
        tempLimit = i
        steps = memoizer(CollatzConjCount, storage)
        if steps > max_steps:
            max_steps = steps
            number_with_max_steps = i
    return number_with_max_steps

#-------------------------------------------------------
def collatz_successor(n):
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1

def build_collatz_graph(limit):
    graph = {}
    for i in range(1, limit + 1):
        successor = collatz_successor(i)
        graph[i] = successor
    return graph

#-----------------Passing Function Arguments----------------------
#Uncomment the line below to answer Question 1
#CollatzConj(num)

#Uncomment the line below to answer Question 2
#counter = CollatzConjCount(num)
#print(f"It takes {counter} steps to reach 1 from {num}")


#Uncomment the line below to answer Question 3 --> CollatzConjCount
```

```
"""start = perf_counter()
memoizedCollatzConjCount = memoizer(CollatzConjCount, storage)
end = perf_counter()
print(f"Time: {end - start}")


start = perf_counter()
CollatzConjCount(num)
end = perf_counter()
print(f"Time: {end - start}")"""

#Uncomment the line below to answer Question 3 --> CollatzConj
"""start = perf_counter()
memoizedCollatzConj = memoizer(CollatzConj, storage)
end = perf_counter()
print(f"Time: {end - start}")


start = perf_counter()
CollatzConj(num)
end = perf_counter()
print(f"Time: {end - start}")"""

#Uncomment the lines below for Question 4
#print(f"Maximum Collatz steps: {max_collatz_steps(limit)+1}")
#print(f"Number with maximum Collatz steps: {max_collatz_num(limit)}")


#Comment the lines below to omit Question 5
graph = build_collatz_graph(limit)
# Print a few entries from the graph for demonstration
for number in range(1, min(limit + 1, 31)):  # Print up to the first 30 numbers
    print(f"{number} -> {graph[number]}")
```