MEMOIZACION

EJERCICIO 1:

1 -Dada una cuadrícula de tamaño m x n, escribí una función que devuelva la cantidad de formas de llegar desde la esquina superior izquierda (0,0) a la inferior derecha (m-1,n-1), moviendo solo hacia la derecha o hacia abajo.

```python
def measure_time(sort_function, arr):
    import time
    start_time = time.time()
    sort_function(arr)
    end_time = time.time()
    return end_time - start_time

def contarCaminosIterativo(m, n):

    matriz = [[1] * n for _ in range(m)]

    for i in range(1, m):
        for j in range(1, n):
            matriz[i][j] = matriz[i - 1][j] + matriz[i][j - 1]

    return matriz[m - 1][n - 1]

def contarCaminosMemo(m, n, memo={}):
    if m == 1 or n == 1:
        return 1

    if (m, n) in memo:
        return memo[(m, n)]

    memo[(m, n)] = contarCaminosMemo(m - 1, n, memo) + contarCaminosMemo(m, n - 1, memo)
    return memo[(m, n)]


def main_ejercicio_1():
    m, n = 255, 255
    print("EJERCICIO 1: Cantidad de caminos en una cuadrícula (255x255)")
    print(f"Tiempo de ejecución (memoizado): {measure_time(lambda x: contarCaminosMemo(m, n), None)} segundos")
    print(f"Tiempo de ejecución (iterativo): {measure_time(lambda x: contarCaminosIterativo(m, n), None)} segundos")

    print(f"Cantidad de caminos (memoizado): {contarCaminosMemo(m, n)}")
    print(f"Cantidad de caminos (iterativo): {contarCaminosIterativo(m, n)}")


if __name__ == "__main__":
    main_ejercicio_1()
```

```
PS C:\Users\Gsu\Documents\PEF> & C:/Python313/python.exe c:/Users/Gsu/Documents/PEF/PRACTICO/memoizacion.py
EJERCICIO 1: Cantidad de caminos en una cuadrícula (255x255)
Tiempo de ejecución (memoizado): 0.05214333534240723 segundos
Tiempo de ejecución (iterativo): 0.008137226104736328 segundos
Cantidad de caminos (memoizado): 29650517331819913093939428137883505420183848762530053263034486796897328586010552773308212988238190828523267214714560499473366568932400654255201713409920
Cantidad de caminos (iterativo): 29650517331819913093939428137883505420183848762530053263034486796897328586010552773308212988238190828523267214714560499473366568932400654255201713409920
```

EJERCICIO 2:

2-Dada una lista de enteros positivos y un número objetivo, escribí una función que determine si existe un subconjunto cuya suma sea exactamente el objetivo.

```python
def existeSubconjuntoIterativo(numeros, objetivo):

    matriz = [False] * (objetivo + 1)
    matriz[0] = True

    for num in numeros:
        for i in range(objetivo, num - 1, -1):
            matriz[i] = matriz[i] or matriz[i - num]

    return matriz[objetivo]

def existeSubconjuntoMemo(numeros, objetivo, n=None, memo={}):
    if n is None:
        n = len(numeros)

    if objetivo == 0:
        return True
    if n == 0:
        return False

    if (n, objetivo) in memo:
        return memo[(n, objetivo)]

    if numeros[n - 1] > objetivo:
        memo[(n, objetivo)] = existeSubconjuntoMemo(numeros, objetivo, n - 1, memo)
    else:
        memo[(n, objetivo)] = (existeSubconjuntoMemo(numeros, objetivo - numeros[n - 1], n - 1, memo) or
                               existeSubconjuntoMemo(numeros, objetivo, n - 1, memo))

    return memo[(n, objetivo)]

def main_ejercicio_2():
    import random
    numeros = [random.randint(1, 100) for _ in range(100)]
    objetivo = 1000
    print("EJERCICIO 2: Existe subconjunto con suma objetivo")
    print(f"Números: {numeros}")
    print(f"Objetivo: {objetivo}")
    print(f"Tiempo de ejecución (memoizado): {measure_time(lambda x: existeSubconjuntoMemo(numeros, objetivo), None)} segundos")
    print(f"Tiempo de ejecución (iterativo): {measure_time(lambda x: existeSubconjuntoIterativo(numeros, objetivo), None)} segundos")
    print(f"Existe subconjunto (memoizado): {existeSubconjuntoMemo(numeros, objetivo)}")
    print(f"Existe subconjunto (iterativo): {existeSubconjuntoIterativo(numeros, objetivo)}")

if __name__ == "__main__":
    #main_ejercicio_1()
    main_ejercicio_2()

```

```
PS C:\Users\Gsu\Documents\PEF> & C:/Python313/python.exe c:/Users/Gsu/Documents/PEF/PRACTICO/memoizacion.py
EJERCICIO 2: Existe subconjunto con suma objetivo
Números: [42, 83, 23, 98, 96, 29, 96, 19, 9, 34, 66, 6, 72, 97, 34, 80, 21, 54, 83, 61, 76, 31, 74, 100, 65, 15, 21, 9, 43, 79, 86, 85, 16, 7
9, 43, 84, 93, 30, 7, 66, 86, 30, 41, 42, 3, 31, 13, 29, 54, 91, 57, 7, 49, 85, 58, 56, 45, 30, 17, 66, 31, 16, 73, 91, 47, 4, 9, 94, 10, 79,
 37, 42, 85, 58, 67, 45, 61, 95, 31, 71, 84, 22, 74, 13, 94, 68, 71, 16, 45, 41, 78, 57, 99, 83, 58, 43, 73, 11, 86, 22]
Objetivo: 1000
Tiempo de ejecución (memoizado): 0.00018835067749023438 segundos
Tiempo de ejecución (iterativo): 0.0055446624755859375 segundos
Existe subconjunto (memoizado): True
Existe subconjunto (iterativo): True
```

EJERCICIO 3

3-Dado un string s y una lista de palabras válidas, determiná si s se puede formar usando solo palabras de la lista, reutilizándolas si es necesario.

```python
def puedeFormarPalabraIterativo(s, palabras):
    n = len(s)
    matriz = [False] * (n + 1)
    matriz[0] = True

    for i in range(1, n + 1):
        for palabra in palabras:
            if matriz[i - len(palabra)] and s[i - len(palabra):i] == palabra:
                matriz[i] = True
                break

    return matriz[n]

def puedeFormarPalabraMemo(s, palabras, memo={}):
    if s == "":
        return True

    if s in memo:
        return memo[s]

    for palabra in palabras:
        if s.startswith(palabra):
            if puedeFormarPalabraMemo(s[len(palabra):], palabras, memo):
                memo[s] = True
                return True

    memo[s] = False
    return False

def main_ejercicio_3():
    s = "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeef"
    palabras = ["e"] * 1000000 + ["f"]
    print("EJERCICIO 3: Puede formar palabra")
    print(f"String: {s}")
    #print(f"Palabras: {palabras}")
    print(f"Tiempo de ejecución (memoizado): {measure_time(lambda x:
puedeFormarPalabraMemo(s, palabras), None)} segundos")
    print(f"Tiempo de ejecución (iterativo): {measure_time(lambda x:
puedeFormarPalabraIterativo(s, palabras), None)} segundos")
    print(f"Puede formar palabra (memoizado): {puedeFormarPalabraMemo(s, palabras)}")
    print(f"Puede formar palabra (iterativo): {puedeFormarPalabraIterativo(s,
palabras)}")

if __name__ == "__main__":
    #main_ejercicio_1()
    #main_ejercicio_2()
    main_ejercicio_3()

```

```
PS C:\Users\Gsu\Documents\PEF> & C:/Python313/python.exe c:/Users/Gsu/Documents/PEF/PRACTICO/memoizacion.py
EJERCICIO 3: Puede formar palabra
String: eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeef
Tiempo de ejecución (memoizado): 0.028084754943847656 segundos
Tiempo de ejecución (iterativo): 0.11107802391052246 segundos
Puede formar palabra (memoizado): True
Puede formar palabra (iterativo): True
```

EJERCICIO 4:

4-Dado un valor total y una lista de monedas, escribí una función que devuelva cuántas combinaciones distintas de monedas suman ese total.

```python
def contarCombinacionesMonedasIterativo(total, monedas):
    combinaciones = [0] * (total + 1)
    combinaciones[0] = 1

    for moneda in monedas:
        for i in range(moneda, total + 1):
            combinaciones[i] += combinaciones[i - moneda]

    return combinaciones[total]

def contarCombinacionesMonedasMemo(total, monedas, n=None, memo={}):
    if n is None:
        n = len(monedas)

    if total == 0:
        return 1
    if total < 0 or n <= 0:
        return 0

    if (total, n) in memo:
        return memo[(total, n)]

    memo[(total, n)] = contarCombinacionesMonedasMemo(total - monedas[n - 1], monedas, n, memo) + \
                       contarCombinacionesMonedasMemo(total, monedas, n - 1, memo)

    return memo[(total, n)]

def main_ejercicio_4():
    total = 90000
    monedas = [1, 5, 10, 25, 50, 100]
    print("EJERCICIO 4: Contar combinaciones de monedas")
    print(f"Total: {total}")
    print(f"Monedas: {monedas}")
    print(f"Tiempo de ejecución (memoizado): {measure_time(lambda x: contarCombinacionesMonedasMemo(total, monedas), None)} segundos")
    print(f"Tiempo de ejecución (iterativo): {measure_time(lambda x: contarCombinacionesMonedasIterativo(total, monedas), None)} segundos")
    print(f"Combinaciones (memoizado): {contarCombinacionesMonedasMemo(total, monedas)}")
    print(f"Combinaciones (iterativo): {contarCombinacionesMonedasIterativo(total, monedas)}")

if __name__ == "__main__":
    #main_ejercicio_1()
    #main_ejercicio_2()
    #main_ejercicio_3()
    main_ejercicio_4()

```

```
PS C:\Users\Gsu\Documents\PEF> & C:/Python313/python.exe c:/Users/Gsu/Documents/PEF/PRACTICO/memoizacion.py
EJERCICIO 4: Contar combinaciones de monedas
Total: 90000
Monedas: [1, 5, 10, 25, 50, 100]
Tiempo de ejecución (memoizado): 0.10027647018432617 segundos
Tiempo de ejecución (iterativo): 0.04295182228088379 segundos
Combinaciones (memoizado): 7915928213224051
Combinaciones (iterativo): 7915928213224051
```