Practical Task 8.2HD

1.

Explanation of the Code and Solution

- For coins it is possible with the help of this Coin Representation algorithm, the number of ways in which a certain total sum of money can be made. In other words, the approach seems to use a combination of memorization and recursion in a manner to save previously computed results for subsequent reusability and to avoid computation for the same input values.

Important Elements of the Solution

- **CoinRepresentation Class:**
  - ✓ The `CoinRepresentation` class contains the essential logic for solving the challenge.
  - ✓ Memorization is used to reduce time to compute seconds by storing the results of particular sums in a cache variable.

- **Solve Method**
  - ✓ This method require recursion and memorization in order to determine the number of possibilities of reaching the total.
  - ✓ Standard Cases: Zero (0) representation(s) if the total is below zero. Return one representation, the empty set, if the total is exactly equivalent to zero.
  - ✓ Utilizing a recursive strategy, the problem is divided into two sub problems: To perform the calculation of sum calculate the sum, and then sum / 2 and sum / 2 - 1 if the amount is even. If the 'sum' is odd, the solution is defined by (sum – 1) / 2.

- ➢ The approach used is recursive in which the solution is broken down into sub-problems, and some results are stored in _cache to avoid computation waste.

Test Generator Class:

- It develops with pre-set results and valid or sound reasons of the test cases. This Generate method offers a set of test inputs, referred to as arg, as well as the results, known as result, which corresponds to these inputs.

Tester Class:

- Checks the functionality of the CoinRepresentation before running the application. Solve the method using the determined test cases from TestGenerator. It records both valid test cases as well as invalid test cases for each test case through the comparison of predicted and actual result. In addition, the assessed amount of time of the test as producing information about how effective the solution is also monitored by the program.

2.

Professional Code Conventions

- **Observation**
  - ✓ The usage of each method is described briefly in the documentation along with the list of input arguments and the return values.

- **Reliability**
  - ✓ However, even in its use in the solution, there is a depending on memorization to enhance efficiency and particularly for larger numbers through caching of results.

- **Reliability in scale**
  - ✓ Which is evident from the test cases shown below, that the program has the ability to accept all sorts of inputs including very big sums.

CoinRepresentation Code

```
using System;
using System.Collections.Generic;

namespace CoinRepresentation
{
   /// <summary>
   ///This class determines how many unique methods there are to use coins to represent a particular total.
   /// For effective computing, it makes use of recursion along with memoization.
   /// </summary>
   public class CoinRepresentation
   {
      // A cache to hold previously calculated outcomes for subproblems
      private static Dictionary<long, long> _cache = new Dictionary<long, long>();

      /// <summary>
      /// Determines how many ways there are to depict a given total.
      /// </summary>
      /// <param name="sum">The target sum to represent.</param>
      /// <returns>how many different methods there are to express the amount.</returns>
      public static long Solve(long sum)
      {
         // Base cases:
         if (sum < 0) return 0;
         if (sum == 0) return 1;

         // Verify if the outcome has previously been cached.
         if (!_cache.ContainsKey(sum))
         {
            if (sum % 2 == 0)
            {

               _cache.Add(sum, Solve(sum / 2) + Solve(sum / 2 - 1));
            }
            else
            {
               // If sum is odd, reduce it and solve for (sum - 1) / 2
               _cache.Add(sum, Solve((sum - 1) / 2));
            }
         }

         // Provide the result that was cached.
         return _cache[sum];
      }
   }
}
```

3.

Pseudocode For Solve

```
function Solve(sum):
  if sum < 0:
    return 0  # No way to represent negative sum

  if sum == 0:
    return 1  # One way to represent sum of 0 (empty set)

  if sum not in cache:
    if sum is even:
      result = Solve(sum / 2) + Solve(sum / 2 - 1)
    else:
      result = Solve((sum - 1) / 2)

    cache[sum] = result  # Store result in cache

  return cache[sum]  # Return cached result if already computed
```

4.

Space and Time Complexity

- **Time Complexity**
  - ✓ The time complexity directly depends on the number of new different sub problems that are created as, due to memorization, each sub problem is solved only once. In terms of time this is O(log n) for a given sum `n` because the count of activation is determined by the logarithm of `n`.

- **Space Complexity**
  - ✓ Since exact cache can contain results for `logn` distinct sums the amount of space taken up by call stack and exact match cache or index contributes to O(log n).

5.

Referenced Materials

- **Kleinberg, J., & Tardos, É. (2005).** *Algorithm Design*. Addison-Wesley.
  - ✓ Using memorisation for extending the utility of recursive calls would not have been conceivable without the conceptual understanding of dynamic programming that forms the basis of this book.

- **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).** *Introduction to Algorithms* (3rd ed.). MIT Press.
  - ✓ Memoization is also discussed in detail as well as recursion together with other algorithms that are covered in this book. The difference was used to find out how best to cache information in a recursive manner.

Online Resources

- **GeeksforGeeks.** *Recursion in C#: How to Solve Problems Using Recursion*. Retrieved from https://www.geeksforgeeks.org
  - ✓ In this article, the recursive decomposition in the Solve method was explained, and the application of recursion in C# to solve problems was shown.

- **Microsoft Documentation.** *Dictionary Class*. Retrieved from https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2
  - ✓ That class in C# can also be used for memoization; the methods for adding, checking, or calling the memoized result are the members of the Dictionary class according to Microsoft.