01286222 / 05506006 Lab 7 Name Nay Chi Shunn Lei id 66011090

Objective(s):

a. To practice problem solving.

b. To understand how to solve problems using recursion, memoization and dynamic programming

Task 1 Create sub package of solutions named pack7_Recursion. Implement **EqualSubsets.java** with following methods

- public static boolean canPartition_Recurse(int [] arr)
- public static boolean canPartition_Memoiz(int [] arr)
- public static boolean canPartition_DP(int [] arr)

The **EqualSubsets** problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is the same.

Example1:

Input -> {1, 5, 11, 5}

Output -> true

Example2:

Input -> {1, 5, 3}

Output -> false

```
public class Lab7 {
    private static void testEqualSubsets() {
      int a [] = {1, 5, 11, 5}
      int b [] = {1, 5, 3}
        System.out.println(EqualSubsets.canPartition_Recur(a));
        System.out.println(EqualSubsets.canPartition_Recur(b));
        System.out.println(EqualSubsets.canPartition_Memoiz(a));
        System.out.println(EqualSubsets.canPartition_Memoiz(b));
        System.out.println(EqualSubsets.canPartition_DP(a));
        System.out.println(EqualSubsets.canPartition_DP(b));
    }
}
```

01286222 / 05506006 Lab 7 Name...Nay Chi Shunn Lei... id ...66011090...

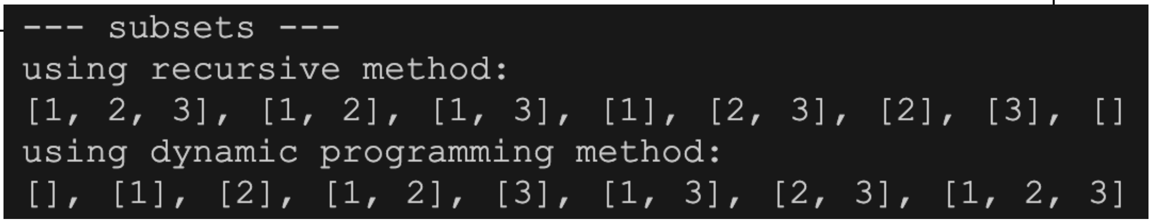
Task 2 Implement **Subsets.java** with following methods

- public static void printAllSubsets_Recurse(List<Integer> set)

- public static void printAllSubsets_DP(List<Integer> set)

The **Subsets** problem is to print all subsets of given set

```
public class Lab7 {
    private static void testSubsets() {
      List<Integer> set = new ArrayList<>();

      set.add(1);

      set.add(2);

      set.add(3);

      println("--- subsets ---");

      println("using recursive method: "

      Subsets.printAllSubsets_Recurse(set);

      println("using dynamic programming method:");

      Subsets.printAllSubsets_DP(set);

    }
```



```
--- subsets ---
using recursive method:
[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []
using dynamic programming method:
[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]
```

What is the time complexity of your algorithm?

There are 2^n possible subsets for a set of size n.
For each subset, the algorithm iterates over the n elements of the set taking O(n) time per subset.
Both the recursive and dynamic programming methods for generating all subsets of a set have a time complexity of $O(n \times 2^n)$.

Task 3 Implement **GridPaths.java** with following method

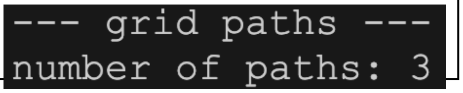
- public static int numberOfPaths(int [][] grid)

The **GridPaths** problem is similar to **Number of Unique Paths** in lecture but with obstacles.

Robot not allowed to move to a space with an obstacle.

An obstacle and a space marked as 1 and 0 respectively in grid.

```
public class Lab7 {
    private static void testGridPaths() {
      int [][] grid = {  {0, 0, 0, 0},

                         {0, 1, 0, 0},

                         {0, 0, 0, 1},

                         {1, 0, 0, 0} };

        println("number of paths: " +

        GridPaths.numberOfPaths(grid));

    }
}
```



```
--- grid paths ---
number of paths: 3
```

What is the time complexity of your algorithm? And try to explain how you calculate it.

The time complexity of the algorithm is O(m×n).

This is because you need to compute the number of paths for each cell in an m×n grid,

and each cell computation takes constant time O(1).

Submission:

EqualSubsets_XXYYYY.java, Subsets_XXYYYY.java and GridPaths_XXYYYY.java and this file.

Due date: TBA