Hands-on Activity 12.1	
Hands-on Activity 12.1 Algorithmic Strategies	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/27/25
Section: CPE21S4	**Date Submitted:** 10/28/25
Name(s): Avila, Vince Gabriel V.	**Instructor:** Engr. Jimlord Quejado

A. Output(s) and Observation(s)

Table 12-1. Algorithmic Strategies and Examples:

Strategy	Algorithm	Analysis
Recursion	**Factorial of a number**	**It solves a problem by repeating the same process on smaller parts until it reaches the simplest case.**
Brute Force	**Guessing a password by trying all combinations**	**It checks every possible answer to find the right one, but it takes a lot of time and effort.**
Backtracking	**Solving a maze**	**It tries different paths and goes back when the way is wrong to find the correct solution.**
Greedy	**Coin Change Problem**	**It picks the best choice at each step to get a quick result, even if it's not always perfect.**
Divide-and-Conquer	**Merge Sort**	**It breaks the problem into smaller parts, solves them one by one, then puts everything together.**

Table 12-2. Memoization Implementation:

Screenshot	`[Memoization] Minimum moves to make 4 become 1: 2`
Analysis	**As the original piece indicates, the number 4 only needs two steps to reach the number 1, which shows that the algorithm efficiently finds the shortest way. The example shows that memoization helps to avoid recalculating so all progress is not lost and time is not wasted.**

Table 12-3. Bottom-Up Dynamic Programming Implementation

Screenshot	`[Iterative DP] Minimum moves to make 4 become 1: 2`

Analysis	**The result shows that it takes 2 steps for 4 to become 1, so the iterative method provides the same correct answer. The iterative method is faster because it builds the solution step-by-step, without using recursion.**

B. Answers to Supplementary Activity

Pseudocode:

```
FUNCTION pathCounter(grid, r, c, sum)
    IF r < 0 OR c < 0 THEN
        RETURN 0
    END IF

    IF r == 0 AND c == 0 THEN
        IF grid[0][0] == sum THEN
            RETURN 1
        ELSE
            RETURN 0
        END IF
    END IF

    RETURN pathCounter(grid, r - 1, c, sum - grid[r][c]) +
           pathCounter(grid, r, c - 1, sum - grid[r][c])
END FUNCTION


MAIN
    grid = [
        [4, 7, 1, 6],
        [6, 7, 3, 9],
        [3, 8, 1, 2],
        [7, 1, 7, 3]
    ]

    targetSum = 25
    totalRows = number of rows in grid
    totalCols = number of columns in grid

    totalPaths = pathCounter(grid, totalRows - 1, totalCols - 1, targetSum)

    DISPLAY "Total paths with sum", targetSum, "=", totalPaths
END MAIN
```

Working C++ Code:

```
#include <iostream>
#include <vector>
using namespace std;

int pathCounter(vector<vector<int>>& grid, int r, int c, int sum) {
    if (r < 0 || c < 0) return 0;
    if (r == 0 && c == 0) return (grid[0][0] == sum) ? 1 : 0;

    return pathCounter(grid, r - 1, c, sum - grid[r][c]) +
           pathCounter(grid, r, c - 1, sum - grid[r][c]);
}
```

```
int main() {
    vector<vector<int>> values = {
        {4, 7, 1, 6},
        {6, 7, 3, 9},
        {3, 8, 1, 2},
        {7, 1, 7, 3}
    };

    int targetSum = 25;
    int totalRows = values.size();
    int totalCols = values[0].size();

    int totalPaths = pathCounter(values, totalRows - 1, totalCols - 1, targetSum);
    cout << "Total paths with sum " << targetSum << " = " << totalPaths << endl;
    return 0;
}
```

Analysis of Working code:

This program determines how many possible routes in a grid may reach a certain overall cost. It makes use of recursion to try all possible ways by moving up or moving towards the left, and subtracting the value in the current cell. I can comprehend how separating the problem into parts as we do worked well to make the puzzle easier to solve.

Screenshot of Demonstration:

```
Total paths with sum 25 = 0


=== Code Execution Successful ===
```

C. Conclusion & Lessons Learned

During this lab, I have learned that different algorithms such as recursion, dynamic programming, and greedy approaches use their own methods to solve problems. I learned that problems are easier to solve when broken down into smaller components and that dynamic programming helps save time when you optimize by recalling results you have already calculated. This activity helped show how planning and mathematical reasoning can simplify difficult problems. I also learned about counting paths in a matrix using recursion, and how doing that affects the running total of costs. All in all, I feel that I did my best, but will need more practice to improve my ability to select the appropriate algorithm.

D. Assessment Rubric

E. External References

W3Schools.com. (n.d.). https://www.w3schools.com/cpp/cpp_functions_recursion.asp

C++ Recursion (With example). (n.d.). https://www.programiz.com/cpp-programming/recursion

GeeksforGeeks. (2025, July 25). *Dynamic Programming or DP*. GeeksforGeeks.

https://www.geeksforgeeks.org/competitive-programming/dynamic-programming