DAA Assignment 6

Aryan Rasiwasia IIT2022024

**Q1.**

**Bruteforce:**

int fib(int n)

{

    if (n <= 1)

        return n;

    return fib(n - 1) + fib(n - 2);

}

int countWays(int s) { return fib(s + 1); }

**Complexity: Time:O(2^n)**

**Space:O(n)**

**Optimized (DP):**

int countWays(int n, vector<int> &dp)

{

    if (n <= 1)

        return dp[n] = 1;

    if (dp[n] != -1) {

        return dp[n];

    }

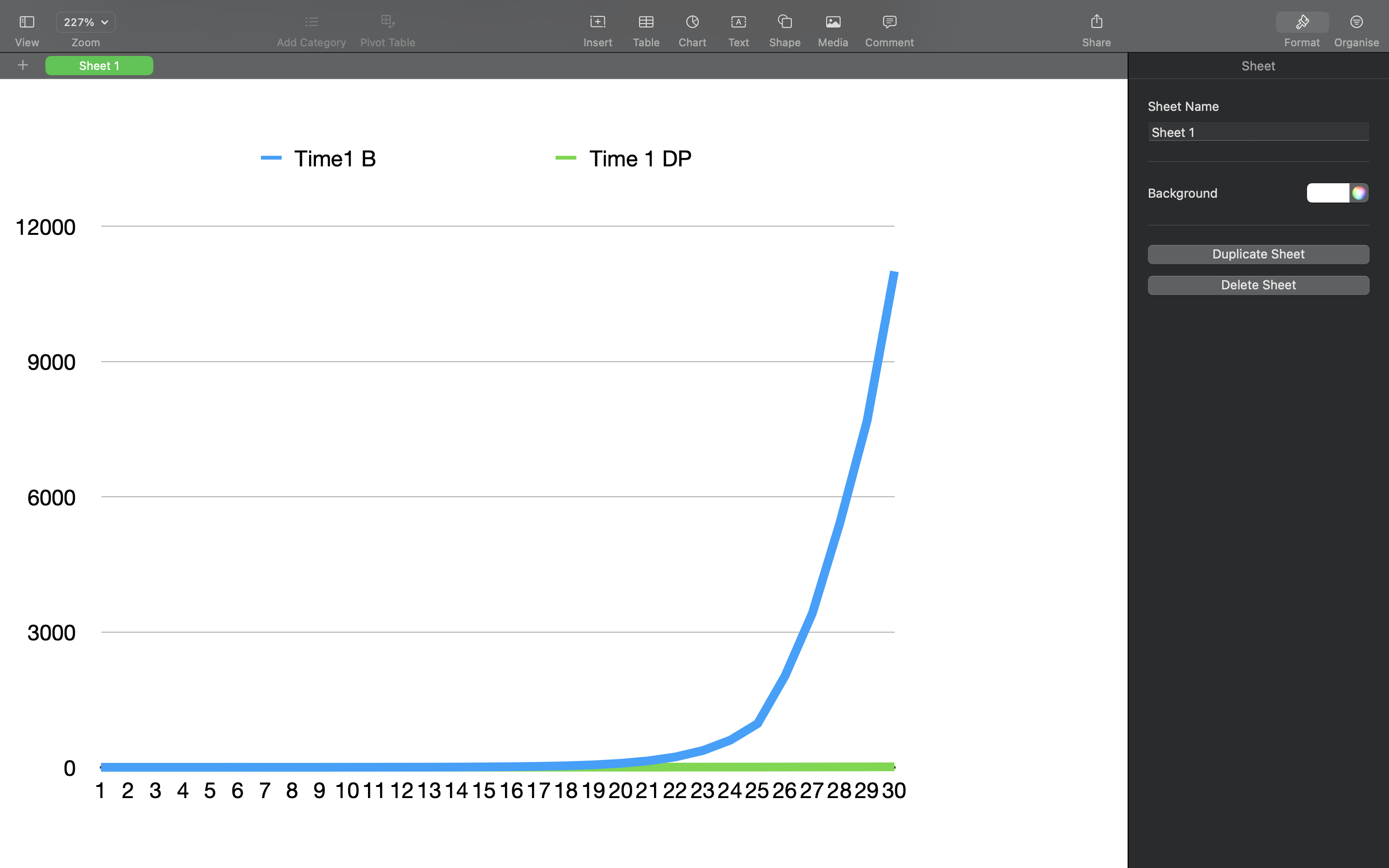
    dp[n] = countWays(n - 1, dp) + countWays(n - 2, dp);

    return dp[n];

}

**Complexity: Time:O(n)**

**Space:O(n)**

****

**Graph:**

**Analysis:**

* —>Bruteforce Approach:
  + In this method, we explore all possible ways of climbing stairs, trying out every combination.
  + It involves a recursive approach, where we consider all possible combinations and calculate the total number of ways to climb stairs.
  + The main drawback is that it can be highly inefficient, especially for larger numbers of stairs, as it recalculates values for the same subproblems multiple times.
* —>Dynamic Programming Approach:
  + DP involves breaking down the problem into smaller subproblems and solving each subproblem only once, storing the solutions to avoid redundant calculations.
  + For climbing stairs, we use a DP table to store the number of ways to climb each step based on the solutions to subproblems.
  + This approach is more efficient as it eliminates redundant calculations, leading to a significant improvement in runtime.

**Q2.**

**Bruteforce:**

int numberOfPaths(int m, int n)

{

       if (m == 1 || n == 1)

       return 1;

    return numberOfPaths(m - 1, n)

           + numberOfPaths(m, n - 1);

    // + numberOfPaths(m-1, n-1);

}

**Complexity: Time:O(2^n)**

**Space:O(n+m)**

**Optimized(DP):**

int numberOfPaths(int n, int m, vector<vector<int>> &DP)

{

    if (n == 1 || m == 1)

        return DP[n][m] = 1;

    // Add the element in the DP table

    // If it was not computed before

    if (DP[n][m] == 0)

        DP[n][m] = numberOfPaths(n - 1, m, DP)

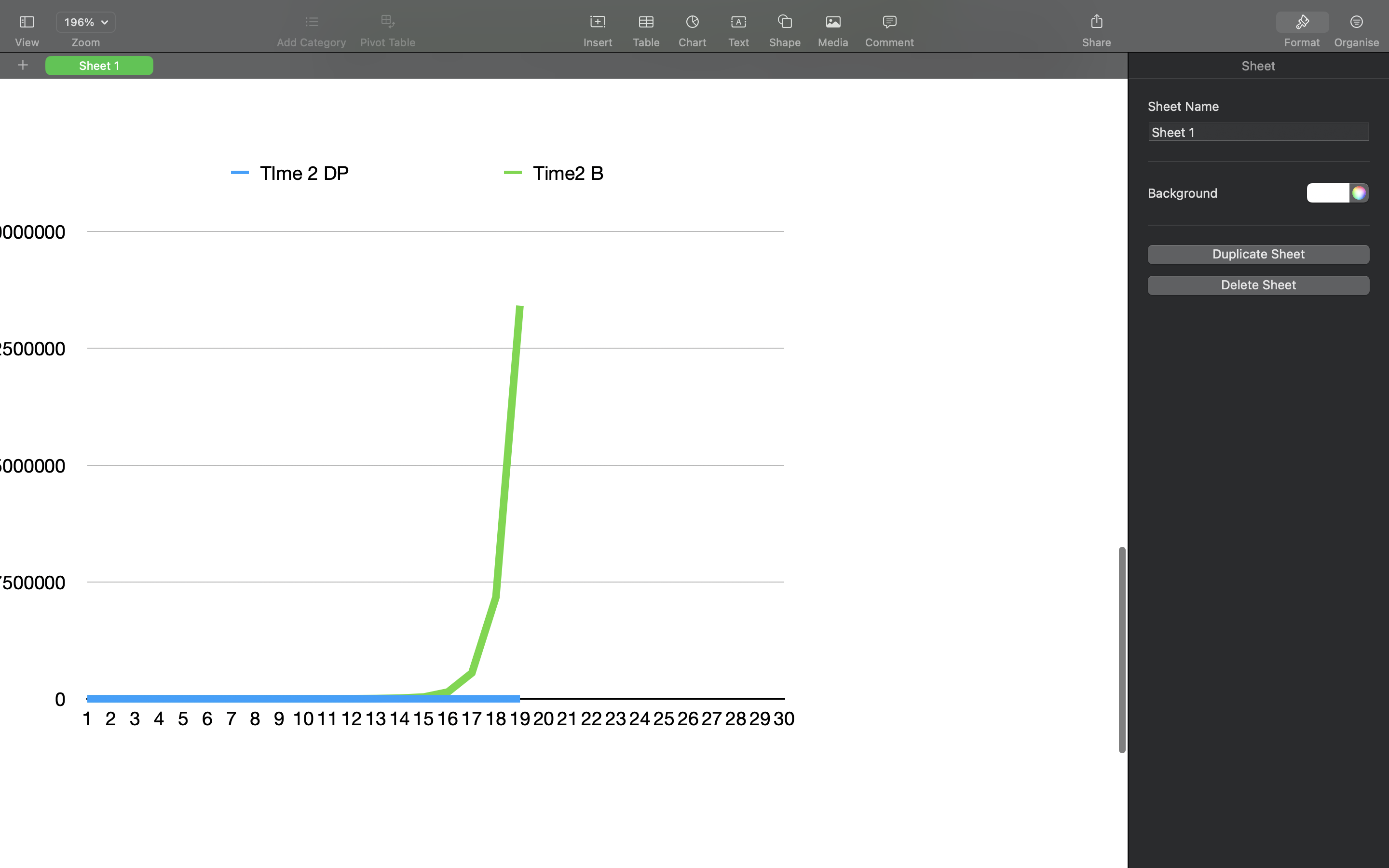
                   + numberOfPaths(n, m - 1, DP);

    return DP[n][m];

}

**Complexity: Time:O(n\*m)**

**Space:O(n\*m)**

****

**Graph:**

**Analysis:**

* —>Bruteforce Approach:
  + In this case, the bruteforce approach involves recursively exploring all possible paths from the top-left corner to the bottom-right corner of a grid.
  + We consider moving either right or down at each step, exploring all combinations until you reach the destination.
  + Similar to the stairs problem, this approach can become inefficient for larger grid sizes due to redundant calculations for the same subproblems.
* —>Dynamic Programming Approach:
  + The DP approach for the unique grid path problem involves building a table to store the number of unique paths to reach each cell in the grid.
  + Starting from the top-left corner, we iteratively fill in the table by summing up the number of paths from the cell above and the cell to the left.
  + By the time we reach the bottom-right corner, the DP table will contain the total number of unique paths, and we avoid recalculating the same subproblems.