**ET3272: Design and Analysis of Algorithms**

**Name of the student: Aditya Jannawar Roll No.: 02**

**Div.: A Batch: 1**

**Date of performance: 13/02/2023**

**Experiment No. 10**

# Title: Binomial Coefficient

**Theory/Description of the Problem Statement:**

The binomial coefficient, denoted as "n choose k" or "C(n, k)", represents the number of ways to choose k objects from a set of n distinct objects, where the order of the objects doesn't matter.

The formula for calculating the binomial coefficient is:

C(n, k) = n! / (k! \* (n-k)!)

where "!" represents the factorial function. For example, 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120.

Alternatively, the binomial coefficient can also be calculated using Pascal's triangle, where each entry in the triangle is the sum of the two numbers directly above it. The first row of the triangle is 1, and each subsequent row is constructed by adding a 1 to the beginning and end of the previous row.

**Algorithm :**

* If k is greater than n, return 0
* If k is equal to 0 or k is equal to n, return 1
* Initialize a variable called "result" to 1
* For each i from 1 to k, do the following:
* Multiply "result" by (n - i + 1)
* Divide "result" by i
* Return the value of "result"

**Pseudo Code :**

* function binomialCoeff(n, k)
* // Initialize a 2D array to store previously computed binomial coefficients
* C = array of size (n+1) x (k+1)
* // Compute and memoize the binomial coefficients
* for i = 0 to n
* for j = 0 to min(i, k)
* if j == 0 or j == i
* C[i][j] = 1
* else
* C[i][j] = C[i-1][j-1] + C[i-1][j]
* // Return the final binomial coefficient
* return C[n][k]

**Analysis of the Algorithm**

The algorithm initializes a 2D array C of size (n+1) x (k+1) to store previously computed binomial coefficients. Then, it computes and memoizes the binomial coefficients using a nested loop. The outer loop iterates from 0 to n, and the inner loop iterates from 0 to min(i, k). For each pair of indices (i, j) in the loop, the algorithm uses the recurrence relation C(i, j) = C(i-1, j-1) + C(i-1, j) to compute the value of C(i, j) and stores it in the 2D array C.

**Time Complexity:**

The time complexity of this algorithm is O(n^2), since we use a nested loop to compute and memoize the binomial coefficients. The space complexity of the algorithm is also O(n^2), since we use a 2D array to store previously computed binomial coefficients.

**Space Complexity:**

The use of dynamic programming results in significant time savings, since the algorithm computes the binomial coefficients once and memoizes the results for later use. This is much faster than the naive approach of computing the binomial coefficient directly using the formula **(n choose k) = n! / (k! \* (n-k)!)**, which can involve repeated computations of factorials.

**Experiment and result:**

Code:

#include <bits/stdc++.h>

using namespace std;

// Returns value of Binomial Coefficient C(n, k)

int binomialCoeffUtil(int n, int k, int\*\* dp)

{

    // If value in lookup table then return

    if (dp[n][k] != -1) //

        return dp[n][k];

    // store value in a table before return

    if (k == 0) {

        dp[n][k] = 1;

        return dp[n][k];

    }

    // store value in table before return

    if (k == n) {

        dp[n][k] = 1;

        return dp[n][k];

    }

    // save value in lookup table before return

    dp[n][k] = binomialCoeffUtil(n - 1, k - 1, dp) +

            binomialCoeffUtil(n - 1, k, dp);

    return dp[n][k];

}

int binomialCoeff(int n, int k)

{

    int\*\* dp; // make a temporary lookup table

    dp = new int\*[n + 1];

    // loop to create table dynamically

    for (int i = 0; i < (n + 1); i++) {

        dp[i] = new int[k + 1];

    }

    // nested loop to initialise the table with -1

    for (int i = 0; i < (n + 1); i++) {

        for (int j = 0; j < (k + 1); j++) {

            dp[i][j] = -1;

        }

    }

    return binomialCoeffUtil(n, k, dp);

}

/\* Driver code\*/

int main()

{

    int n = 5, k = 2;

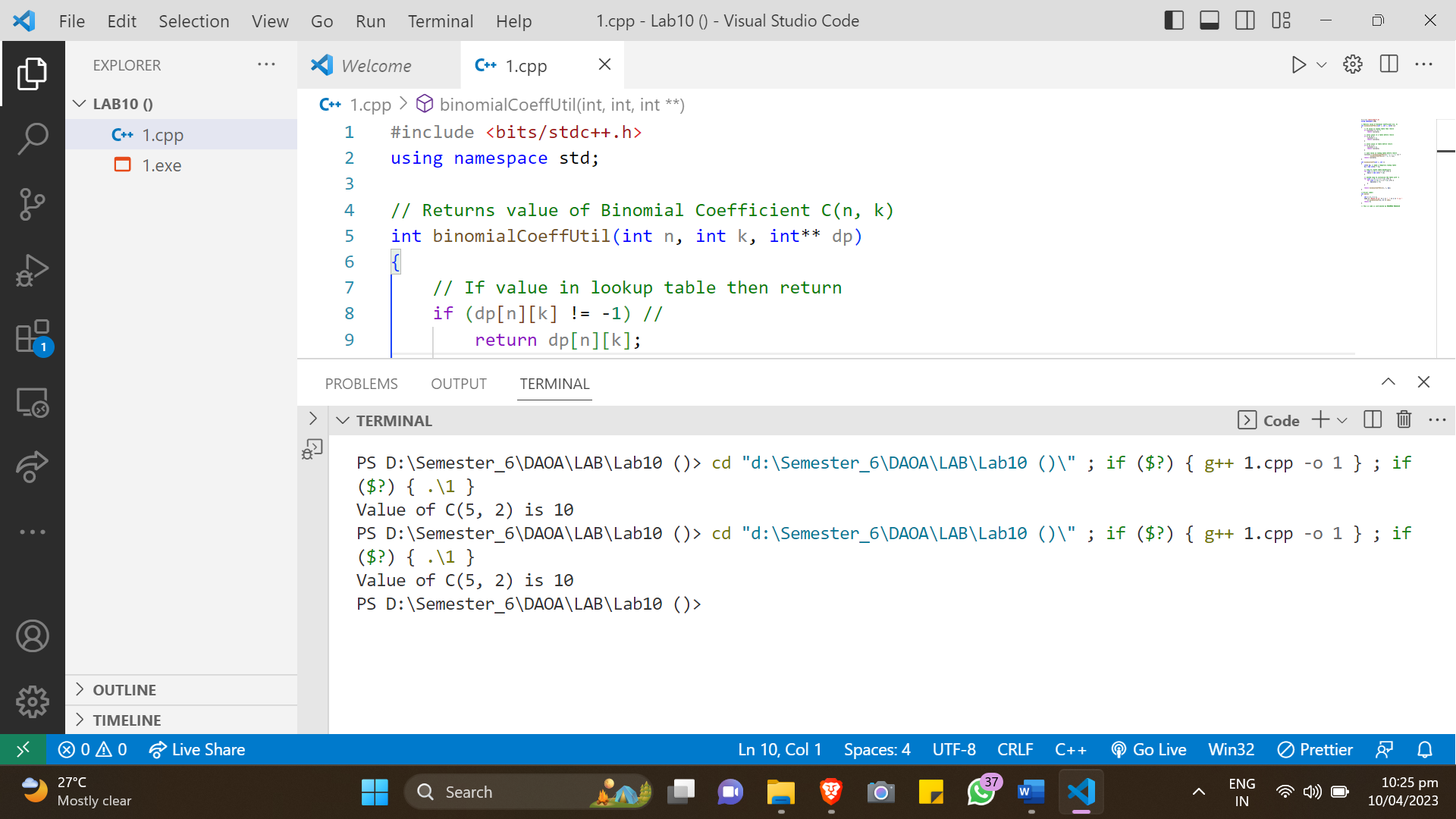
    cout << "Value of C(" << n << ", " << k << ") is "

        << binomialCoeff(n, k) << endl;

    return 0;

}

Output:



**Conclusions:**

In conclusion, the algorithm for computing binomial coefficients using dynamic programming is a fast and efficient solution that can handle large values of n and k with reasonable time and space requirements. The algorithm memoizes previously computed values to avoid redundant calculations, resulting in significant time savings compared to the naive approach of computing the binomial coefficient directly using factorials. The time complexity of the algorithm is O(n^2), and the space complexity is also O(n^2). Overall, this algorithm is an excellent example of how dynamic programming can be used to optimize a problem and improve its efficiency.