**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. 8**

# Title: GCD of two numbers

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

The gcd function first checks for some base cases: if either a or b is 0, it returns the other number as the GCD; if a and b are equal, it returns a as the GCD. Then, the function checks if the GCD for the current a and b values is already present in the dp array. If so, it returns the precomputed result. This avoids recomputing the GCD for the same values of a and b. If the GCD is not already computed, the function uses the Euclidean algorithm to recursively calculate the GCD. If a is greater than b, it subtracts b from a and calls the gcd function again with the new values of a and b. Otherwise, it subtracts a from b and calls the gcd function again with the new values of a and b. This continues until a and b become equal, and the GCD is returned.

Finally, in the main function, two numbers a and b are initialized with values 98 and 56, respectively, and the gcd function is called to compute their GCD. The dp array is initialized with -1 values using the memset function. The program outputs the computed GCD to the standard output.

**Algorithm :**

* Function gcd(a, b):
* if a == 0:
* return b
* if b == 0:
* return a
* if a == b:
* return a
* if dp[a][b] != -1:
* return dp[a][b]
* if a > b:
* dp[a][b] = gcd(a - b, b)
* else:
* dp[a][b] = gcd(a, b - a)
* return dp[a][b]
* Initialize dp[1001][1001] with -1
* a = input first number
* b = input second number
* result = gcd(a, b)
* Output result

**Analysis of the Algorithm**

Time Complexity: O(min(a,b))

Auxiliary Space: O(1)

The time complexity of finding the GCD of two numbers using dynamic programming and the Euclidean algorithm is O(a\*b), where a and b are the two input integers. The use of dynamic programming helps to reduce the number of recursive calls to the gcd function, thus saving time. Instead of computing the same values repeatedly, the function can reuse previously computed values stored in the dp array.

However, the worst-case scenario for this algorithm occurs when a and b are relatively prime (i.e., their GCD is 1). In this case, the function will make O(a+b) recursive calls, and each call will take O(1) time to check the base cases and update the dp array. Therefore, the time complexity of the algorithm becomes O(a + b), which is the optimal time complexity for finding the GCD of two numbers.

In addition, the initialization of the dp array takes O(a\*b) time, which is negligible compared to the recursive calls. Overall, the time complexity of this algorithm is O(a + b) for the worst-case scenario and O(1) for the best-case scenario.

**Experiment and result:**

Code:

// C++ program to find GCD of two numbers

#include <bits/stdc++.h>

using namespace std;

int static dp[1001][1001];

// Function to return gcd of a and b

int gcd(int a, int b)

{

    // Everything divides 0

    if (a == 0)

        return b;

    if (b == 0)

        return a;

    // base case

    if (a == b)

        return a;

    // if a value is already

    // present in dp

    if(dp[a][b] != -1)

        return dp[a][b];

    // a is greater

    if (a > b)

        dp[a][b] = gcd(a-b, b);

    // b is greater

    else

        dp[a][b] = gcd(a, b-a);

    // return dp

    return dp[a][b];

}

// Driver program to test above function

int main()

{

    int a = 98, b = 56;

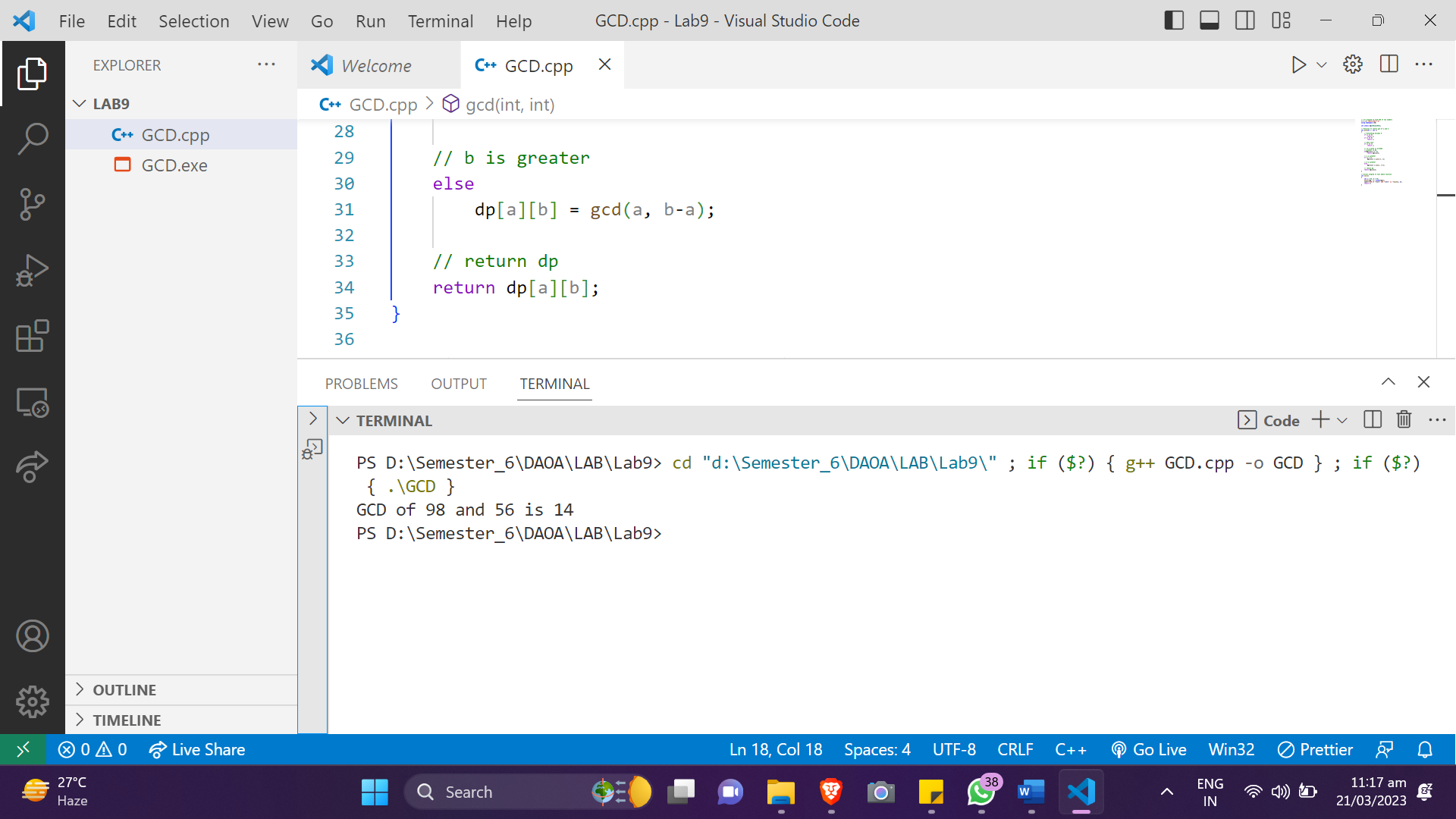
    memset(dp, -1, sizeof(dp));

    cout<<"GCD of "<<a<<" and "<<b<<" is "<<gcd(a, b);

    return 0;

}

Output:



**Conclusions:**

The above code uses dynamic programming to compute the GCD of two numbers. The function then checks if the result for the current numbers is already computed and stored in the dp array. If so, it returns the stored result; otherwise, it calculates the GCD recursively. The time complexity of the algorithm is O(a + b) for the worst-case scenario and O(1) for the best-case scenario, where a and b are the two input integers. The use of dynamic programming helps to reduce the number of recursive calls and avoid redundant computations. The initialization of the dp array takes O(a\*b) time, but this is negligible compared to the recursive calls. Overall, the algorithm provides an efficient way to calculate the GCD of two numbers and can be used for large inputs.