NAME: Atharva Kangralkar

ROLL NO: 54

CLASS: CS(AIML)-A

PRN NO: 12311493

BATCH: 2

LAB ASSIGNMENT 7

AIM: Compare dynamic programming and divide and conquer using Fibonacci sequence

STEP 1: CODE

dynamic programming:-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
   int n;
   cout << "Enter n: ";
   cin >> n;

   vector<long long> dp(n+1);  // DP array
   dp[0] = 0;
   dp[1] = 1;

   for (int i = 2; i <= n; i++) {
      dp[i] = dp[i-1] + dp[i-2];
   }

   cout << "Fibonacci(" << n << ") = " << dp[n] << endl;
   return 0;
}
```

OUTPUT:

```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ fibonacci_DP_bottom_up.cpp -o fibonacci_DP_bottom_up } ; if ($?) { .\fibonacci_DP
_bottom_up }
Enter n: 6
Fibonacci(6) = 8
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ fibonacci_DP_bottom_up.cpp -o fibonacci_DP_bottom_up } ; if ($?) { .\fibonacci_DP
_bottom_up }
Enter n: 3
Fibonacci(3) = 2
PS C:\TY CS(AIML)\DAA\Lab Codes>
```

Code:

divide and conquer:-

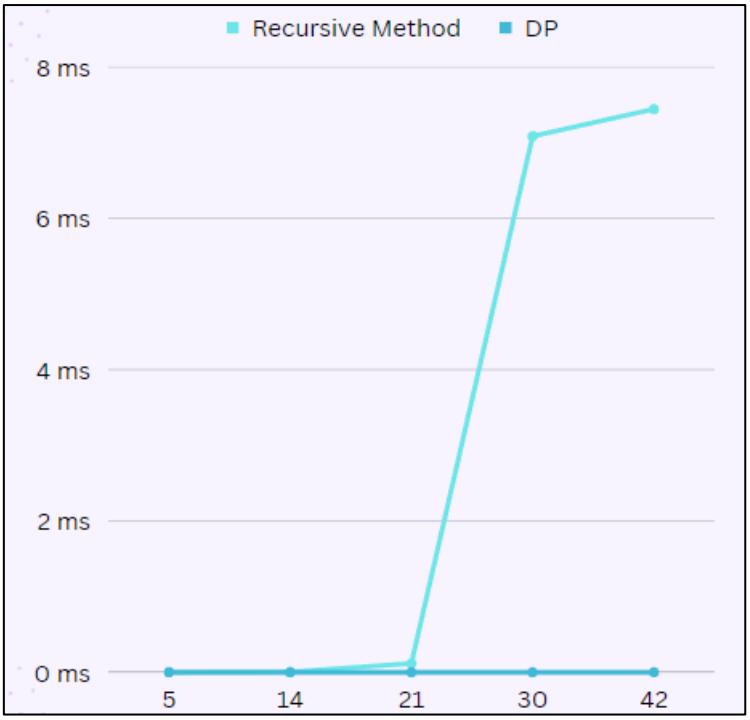
```
#include <bits/stdc++.h>
using namespace std;

// Recursive function to find nth Fibonacci number
long long fib(int n) {
   if (n <= 1)   // base cases
      return n;
   return fib(n-1) + fib(n-2); // recursive relation
}

int main() {
   int n;
   cout << "Enter n: ";
   cin >> n;

   cout << "Fibonacci(" << n << ") = " << fib(n) << endl;
   return 0;
}
```

OUTPUT:

```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ fibonacci_recursive.cpp -o fibonacci_recursive } ; if ($?) { .\fibonacci_recursiv
e }
Enter n: 6
Fibonacci(6) = 8
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ fibonacci_recursive.cpp -o fibonacci_recursive } ; if ($?) { .\fibonacci_recursiv
e }
Enter n: 2
Fibonacci(2) = 1
```

STEP 2: COMPARISON

n	Execution time recursive	Execution time DP
5	0.000003	0.000003
10	0.000010	0.000003
20	0.000917	0.000003
30	0.130733	0.000011
35	1.471457	0.000015



Time Complexity analysis

RECURSSIVE METHOD

The time complexity of the recursive method to find the nth Fibonacci number is O(2^n), which means that the time taken to compute the Fibonacci number increases exponentially with n. This is because each recursive call generates two more recursive calls until it reaches the base case of n=0 or n=1. As a result, the number of function calls grows exponentially with n, leading to an exponential time complexity.

However, if we use memorization to store the results of previously computed Fibonacci numbers, we can reduce the time complexity to O(n), which is linear. This is because we only need to compute each Fibonacci number once, and then we can reuse the stored results for subsequent computations.

DYNAMIC PROGRAMMING

The time complexity of the dynamic programming approach to find the nth Fibonacci number is O(n), which is linear.

In this approach, we start by initializing an array with the first two Fibonacci numbers (0 and 1). We then loop through the remaining indices of the array, computing each Fibonacci number as the sum of the two previous numbers.

By using this approach, we avoid the repeated computations that occur in the recursive approach, which leads to an exponential time complexity. Instead, we compute each Fibonacci number once and store the result in the array, which allows us to reuse the computed values in subsequent computations.

As a result, the time complexity of the dynamic programming approach is linear, proportional to the input size of n, making it more efficient than the recursive approach.