USING DYNAMIC PROGRAMING TO SOLVE FIBONACCI

ASSIGNMENT 2

ADVANCED ALGORITHMS

COURSE CODE: CSC 811

STUDENT NAME: EZE ALLOYSIUS IZUCHUKWU

MATRIC NUMBER: 24PGCD00498

LECTURER: DR. F. ASANI

LANDMARK UNIVERSITY

DATE: 20TH OF January, 2025

INTRODUCTION

There are fundamentally two methods of solving Fibonacci using Dynamic programing, the two methods includes Memoization and Tabulation:

The Fibonacci numbers is given in the sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Each number in the sequence is gotten by adding up the two numbers before it.

It can also be written as written 1, 1, 2, 3, 5, 8, 13, 21, 34 … this will be used where fib(0) = 0.

This function will return the nth Fibonacci number.

```
Function fibonacciNoRecursion(x){
  if (x < 0) return undefined;
  if (x === 0) return 0;
  let previous = 1;
  let sum = 1;
  for (let i = 2; i <= x; i++){
    let temp = sum;
    sum += previous;
    previous = temp;
  }
  return sum
}
```

This has a linear time complexity — O(x)

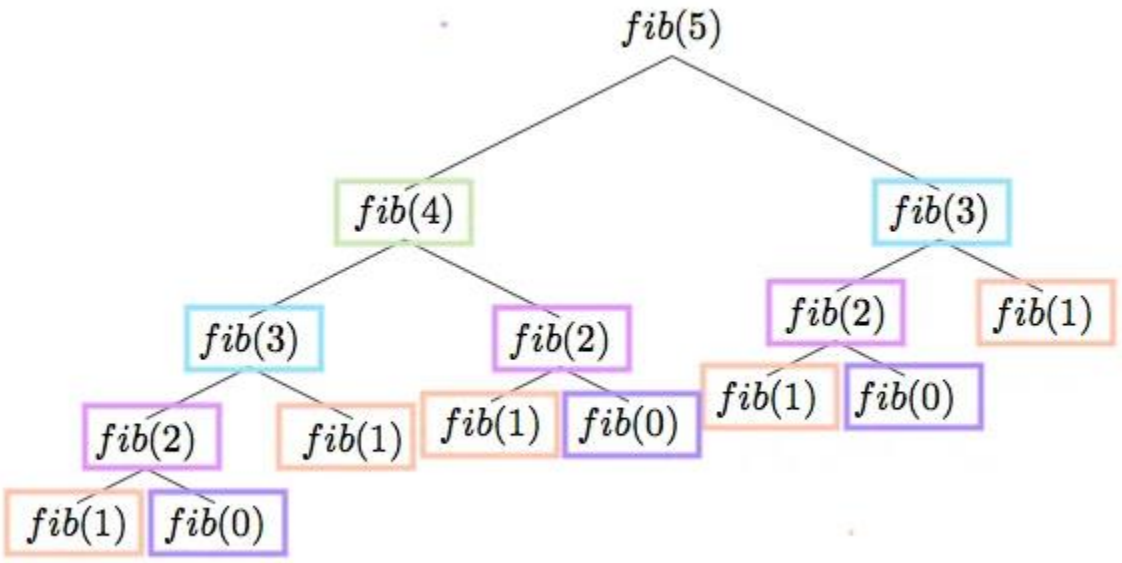
Here is the Recursive solution:

```
function fib(x){
  if (x< 0) return undefined;
  if (x < 2) return x;
  return fib(x-1) + fib(x-2)
}
```

But its time complexity is exponential, or O(2^x), which is not ideal at all.

In the case of **Overlapping Subproblems**

Subproblems are smaller versions of the original problem:



There are two ways to solve the Fibonacci problem using dynamic programming. They are Memoization and Tabulation:

1. Memoization

Memoization stores the result of expensive function calls and returns the stored results whenever the same inputs occur again. This enables to remember any values we have already calculated and access them instead of repeating the same calculation.

This approach is a top-down approach, which implies that we start with what we are trying to find. We start from fib(5) and work our way down, filling in the gaps and adding them all together. (To to find fib(5) we calculate fib(4) and fib(3) etc.)

```
Function memoizedFib(x, memo={}){
  if (memo[x]){
    return memo[x];
```

```
  }
  else if (x === 0 || x === 1){
    return 1;
  }
  let result = memoizedFib(x-1, memo) + memoizedFib(x-2, memo);
  memo[x] = result;
  return result;
}
```

2. Tabulation

Tabulation is accomplished through iteration (a loop). If we start from the smallest subproblem, we could store the results in a table (an array), do something with the data (add the data for Fibonacci) until we get to the solution.

This approach is called a bottom-up approach. We start from the bottom, finding fib(0) and fib(1), add them together to get fib(2) and so on until we reach fib(5).

```
function tabulatedFib( x ){
  if ( x === 1 || x === 2){
    return 1;
  }
  const fibNums = [0, 1, 1];
  for (let i = 3; i <= x; i++){
    fibNums[i] = fibNums[i-1] + fibNums[i-2];
  }
  return fibNums[ x ];
}
```

The time complexity of memoization and tabulation solutions are O(x) — time grows linearly with the size of x, because we are calculating fib(4), fib(3), etc each one time.

Note: While both have the linear time complexity, since memoization uses recursion, if you try to find the Fibonacci number for a large x, you will get a stack overflow (maximum call stack size exceeded).

Hence, Tabulation, on the other hand, doesn't take as much space, and will not cause a stack overflow.