Lab 13
The Fibonacci Sequence

Defined mathematically as:

\[
F_n \begin{cases} 
1 & \text{if } n = 1, 2 \\
F_{n-1} + F_{n-2} & \text{if } n \geq 3
\end{cases}
\]

Each \( n \)th Fibonacci number is the sum of the previous two Fibonacci numbers, with the exception of the first two numbers, which both start at 1. Now in C++:

```cpp
int fib(int n) {
    if (n == 1 || n == 2) return 1;
    return fib(n-1) + fib(n-2);
}
```
The Problem

The repeated calculations causes the program to make an exponential amount of recursive calls, and therefore, result in exponential run-time.
Bottom-Up

Start from smaller cases and store the calculated values in a table for future use, gradually approach to the final solution.

Cache the result in a loop up array to avoid repeating calculation. To maintain the accessibility and without recreating a new one, there are two ways:

Use of global variable: Global variables are accessible from anywhere in the program, this could make the code harder to manage.

Use of static variable: Static variable are accessible from the function in which it is declared, however the memory space it takes up does not get recreated each time the function is called.
Solution – Memoize it!

```plaintext
int fib(int n) {
    static int memo[100] = {0}; //L1
    if (n == 1 || n == 2) return 1;
    if (memo[n] != 0) //L2
        return memo[n];
    return memo[n] = fib(n-1) + fib(n-2); //L3
}
```

Variables declared in a function normally “disappear” after the function terminates. The static keyword keeps a variable “alive” for the duration of the program. In other words, memo will not be re-declared/reinitialized upon a subsequent call to the fib function.

L1: Declaring a static integer array of size 100, initialized to all zeroes
L2: If memo[n] is not 0, fib(n) has been calculated already, so just return the stored result!
L3: Otherwise, calculate and store result of recursive call, then return it.
Memoization helps us avoid repeated calculations by simply returning already-calculated values (stored in the memo array). This helps give the recursive Fibonacci function a linear time complexity.
int fib(int n) {
    int *memo = new int[n+1];
    for (int i = 0; i <= n; ++i)
        memo[i] = 0;
    int r_val = _fib(n, memo);
    delete []memo;
    return r_val;
} //or use STL maps!

int _fib(int n, int memo[]) {
    if (n == 1 || n == 2) return 1;
    if (memo[n] != 0)
        return memo[n];
    return memo[n] = _fib(n-1, memo) + _fib(n-2, memo);
} //helper function