Intro to Dynamic Programming

Problem 1: Fibonacci

The Fibonacci function is defined as follows:

$$\begin{aligned} F(0) &= 1 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ for } n > 2 \end{aligned}$$

(a) Implement a recursive procedure for computing the Fibonacci sequence based directly on the function defined above.

(b) Then the running time of this algorithm can be expressed as:

$$T(n) = T(n-1) + T(n-2) + 1$$

Choose from the following asymptotic bounds the one that best satisfies the above recurrence and explain your selection:

i $T(n) = O(n)$

ii $T(n) = O(n^2)$

iii $T(n) = \Omega(c^n)$, for some constant c

iv $T(n) = \Theta(n^n)$

(c) What specifically is wrong with your algorithm? (i.e., what observation can you make to radically improve its running time?)

(d) The memoized version is based on the following recurrence: $F[i] = F[i-1] + F[i-2]$. (Which is the same recurrence as stated above.) However, filling in the solution with memoization, we shortcut the direct recursive approach by performing a table lookup before calling a new subproblem. Help complete the memoized recursive algorithm for computing $F(n)$ efficiently.

```
MemoFib(n){
   F[0] = F[1] = 1
   if F[i-1] = empty{
      //??

   }
   if F[i-2] = empty{
      //??

   }
   return F[i-1] + F[i-2]
}
```

(e) Why is the runtime of the algorithm `MemoFib` $O(n)$.

(f) Here is a traditional bottom-up dynamic programming algorithm for computing $F(n)$ efficiently. What is the runtime of `DynamicFib`?

```
DynamicFib(n){
   if (n = 0) { return 1 }
   Fib_previous = 1
   Fib_current = 1
   for i = 2 to n{
      Fib_new = Fib_previous + Fib_current
      Fib_previous = Fib_current
      Fib_current = Fib_new
   }
   return Fib_current
}
```

(g) How does this algorithm work? We calculate the smaller values of `Fibonacci` first, then build larger values from them. What tradeoff is in the favor of `DynamicFib` compared to `MemoFib`?