Assignment 3
Name: Ataias Pereira Reis ID: 260590875
Name: Ahmad Saif ID: 260329527

Certification "We certify that both partners contributed equally to the work submitted here and that it represents solely our own efforts"

Question 3

Poormemoizer The Fibonacci function was defined as a recursive function, so this means it call itself. Our written function to memoize Fibonnaci called the memoized function everytime a recursive call was made, and if the value was memoized, than it was retrieved, otherwise, computed the missing values up to the number of the sequence we want to obtain, and saving the results. In the case of poormemoizer, the recursive calls are made to the original function, not the memoized function. fib(60), for example, needs fib(59) and fib(58). If the function created in question 1 was run with fib(59), then fib(60) would be easily computed as fib(59) + fib(58), and then saved in memory. In the poormemoized function, even if it was run with fib(59) before fib(60) (or a greater number in the sequence), it ignores it and computes from scratch everytime, as fib(59) + fib(58) + fib(57) + . . . + fib(0). This makes things much slower.

Memoizer and side-effect function Memoizer is great for functions that are in a model similar to the Fibonnaci, recursive or non-recursive functions, that return a specific value for an input. In the case of functions that have side-effects, as printing something, and printing is the only side-effect, then the memoizer would just be memoizing unit() as what the function does, and this wouldn't be helpful for anything.

In the case of returning a value, ignoring side-effects, if they do not influence the value of the output, then it is useful to use this memoizer, as the core computations will be memoized. In the case of the side-effects influencing the output, then this would be a mess. The output would consider the state of mutable variables in the first call, and then just give the same output again, even though the result should be different. An example of this follow below:

```
let x = ref 0
let rec f n =
  x := !x + 1; !x + n
```

In this code we have a strong depence of the state to obtain the final result of the function, and our memoizer doesn't help in this cases, but will change deeply the function and give wrong results.

Question 4