python

```
#####def fibonacci_memo(n, memo={}):
  if n in memo:
    return memo[n]
  if n <= 1:
    return n
  memo[n] = fibonacci_memo(n - 1, memo) + fibonacci_memo(n - 2, memo)
  return memo[n]

# Example usage
n = 30
print(f"Fibonacci of {n} is {fibonacci_memo(n)}")###  # Output: Fibonacci of 30 is 832040
```

Explanation:.

- **Time Complexity: The time complexity is reduced to O(n)O(n)O(n) with memoization because each Fibonacci number is computed only once.**

Summary

- **Exponential Complexity: Algorithms with O(2n) O(2^n) O(2n) complexity can be very slow for larger inputs due to their rapid growth in the number of operations.**
- **Improvement: Techniques like memoization or dynamic programming can often transform an O(2n) O(2^n) O(2n) algorithm into a more efficient O(n)O(n)O(n) or O(n2)O(n^2) O(n2) algorithm.**

****

% formatting in python

1. **%s**: String
2. **%d**: Integer (decimal)
3. **%f**: Floating-point number
4. **%x**: Hexadecimal (lowercase letters)
5. **%X**: Hexadecimal (uppercase letters)
6. **%o**: Octal
7. **%e**: Exponential notation (lowercase e)
8. **%E**: Exponential notation (uppercase E)