

# Time vs Space Complexity



in DSA, do we only concern ourselves with Time Complexity? Is space complexity not important?

Both **time complexity** and **space complexity** are important in **Data Structures and Algorithms (DSA)**.

- **Time Complexity** tells us how fast an algorithm runs as the input size increases.
- **Space Complexity** tells us how much extra memory an algorithm uses as the input size increases.

While **time complexity** is often prioritized because we want faster solutions, **space complexity** matters too, especially in memory-constrained environments. Some algorithms optimize one at the expense of the other, so a balance is needed.

For example:

- **Merge Sort** has a time complexity of  $O(n \log n)$  but requires  $O(n)$  extra space.
- **Quick Sort** also has  $O(n \log n)$  time complexity on average but uses  $O(\log n)$  extra space.
- **Dynamic Programming** solutions often reduce time complexity but increase space complexity due to memoization.

So, **both time and space complexities should be considered** when analyzing algorithms. Some problems prioritize speed, while others prioritize memory efficiency.



what is "memoization"?

**Memoization** is an optimization technique used to improve the efficiency of algorithms, particularly in **recursive** problems. It involves storing the results of **previously computed function calls** and reusing them instead of recalculating them. This helps in **avoiding redundant computations**, significantly improving performance.

**Memoization** is an optimization technique used in **Dynamic Programming (DP)** where we store the results of expensive function calls and reuse them when needed instead of recomputing them. This helps reduce **time complexity** by avoiding redundant calculations.

## How It Works:

1. Check if the result of a subproblem has already been computed.
2. If it is, return the stored result (cached value).
3. If not, compute the result, store it, and return it.

## Example of Memoization:

Let's consider the **Fibonacci sequence**:

Without memoization (**plain recursion**):

```
python

def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2) # Recomputes values multiple times

print(fib(5)) # 0, 1, 1, 2, 3, 5
```

1/2

With memoization (**using a dictionary to store results**):

```
python

def fib_memo(n, memo={}):
    if n in memo:
        return memo[n] # Return already computed result
    if n <= 1:
        return n
    memo[n] = fib_memo(n - 1, memo) + fib_memo(n - 2, memo)
    return memo[n]

print(fib_memo(5)) # 5
```

- **Time Complexity:  $O(n) \rightarrow$  Linear**
- It **stores previously computed values**, avoiding redundant calculations.

## Key Takeaways:

- **Used in recursion-heavy problems** (Fibonacci, Dynamic Programming).
- **Reduces time complexity** from exponential ( $O(2^n)$ ) to polynomial or linear ( $O(n)$ ).
- **Commonly implemented using dictionaries (hash tables) or arrays.**
- **Alternative approach: **Tabulation** (Bottom-Up DP)** – solving problems iteratively instead of recursively.

Would you like me to show a different example, maybe for a different problem?

ChatGPT can make mistakes. Check important info.



## Example: Fibonacci Sequence (Without Memoization)

A simple recursive Fibonacci function has  **$O(2^n)$  time complexity** because it recomputes values multiple times.

OpenAI

```
python

def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2) # Recomputes values unnecessarily
```

## Example: Fibonacci with Memoization

Using **memoization**, we store computed values to avoid redundant calculations, reducing the time complexity to  **$O(n)$** .

```
python

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

## Key Points:

- Used mainly in **top-down Dynamic Programming** (recursion + caching).
- Avoids repeated calculations and improves efficiency.
- Can be implemented using **dictionaries (Python), arrays, or hash tables**.

Would you like an example related to your exam syllabus?

ChatGPT can make mistakes. Check important info.