




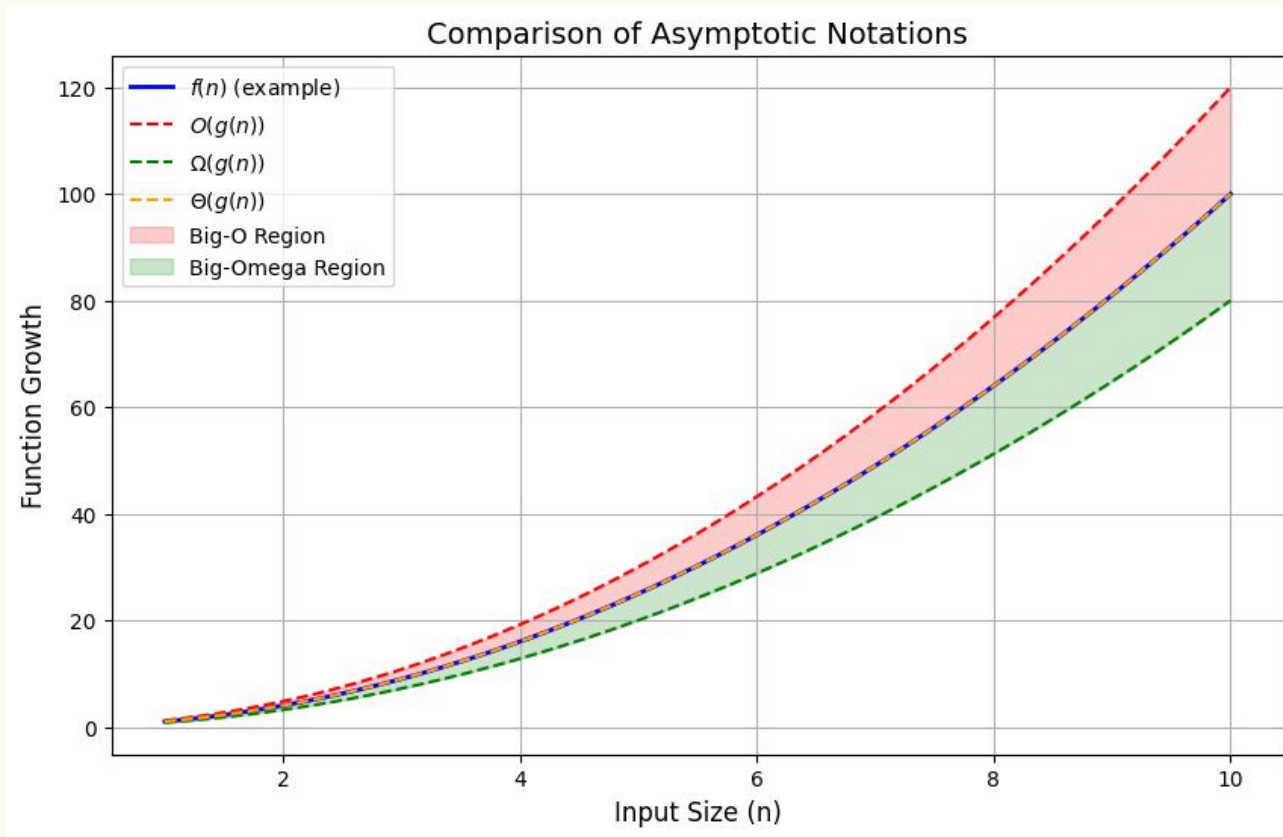
# Asymptotic Notation & DP

CSE 2202 - Design & Analysis of  
Algorithms

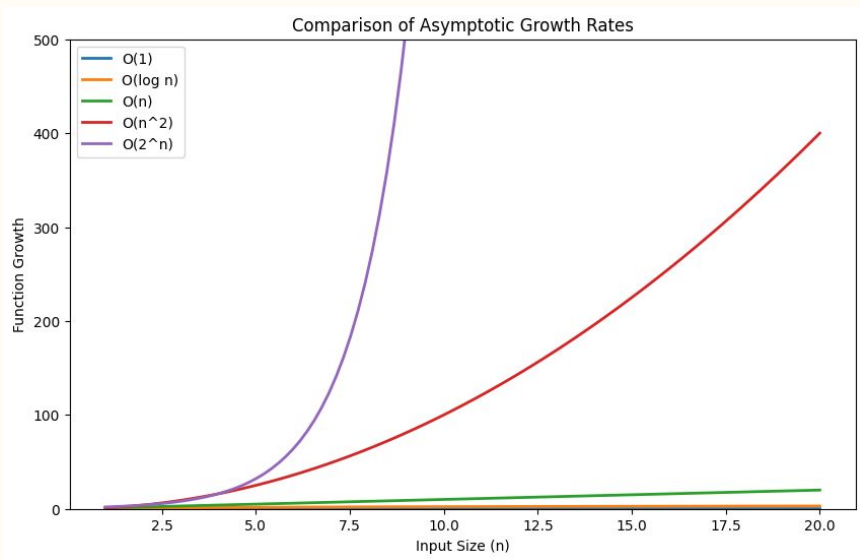


We use **Asymptotic Notations** to describe the **performance** of an algorithm

# Asymptotic Notations



# Complexity



$O(1)$  - Constant

$O(\log n)$  - Logarithmic

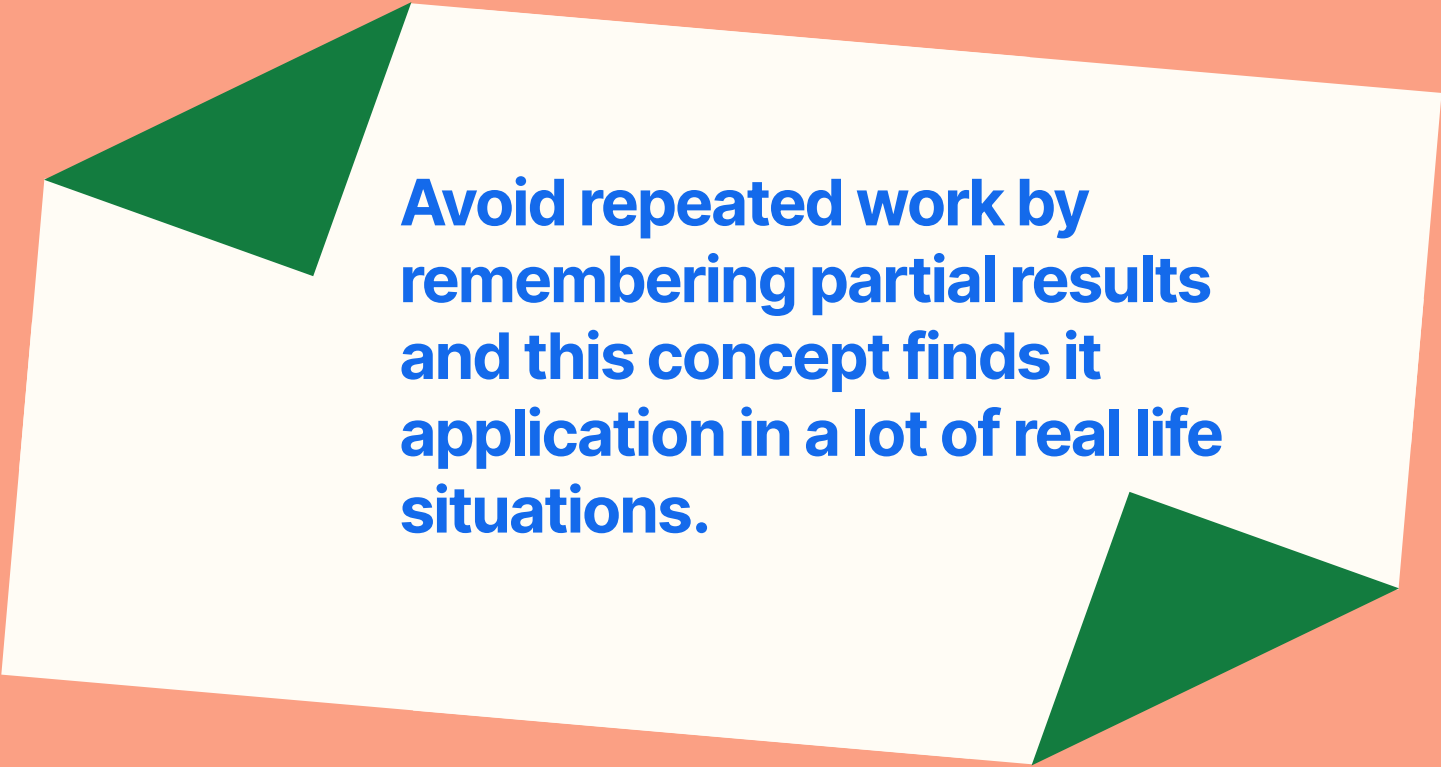
$O(n)$  - Linear

$O(n^2)$  - Quadratic

$O(2^n)$  - Exponential

Those who cannot remember the past  
are condemned to repeat it.

-Dynamic Programming



**Avoid repeated work by remembering partial results and this concept finds it application in a lot of real life situations.**

# What is DP?

[Jonathan Paulson](#) explains Dynamic Programming in his amazing Quora answer [here](#).

Writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper.

"What's that equal to?"

Counting "Eight!"

Writes down another "1+" on the left.

"What about that?"

"Nine!" " How'd you know it was nine so fast?"

"You just added one more!"

"So you didn't need to recount because you remembered there were eight! Dynamic Programming is just a fancy way to say remembering stuff to save time later!"

# What is Dynamic Programming

Dynamic programming is basically, recursion plus using common sense.

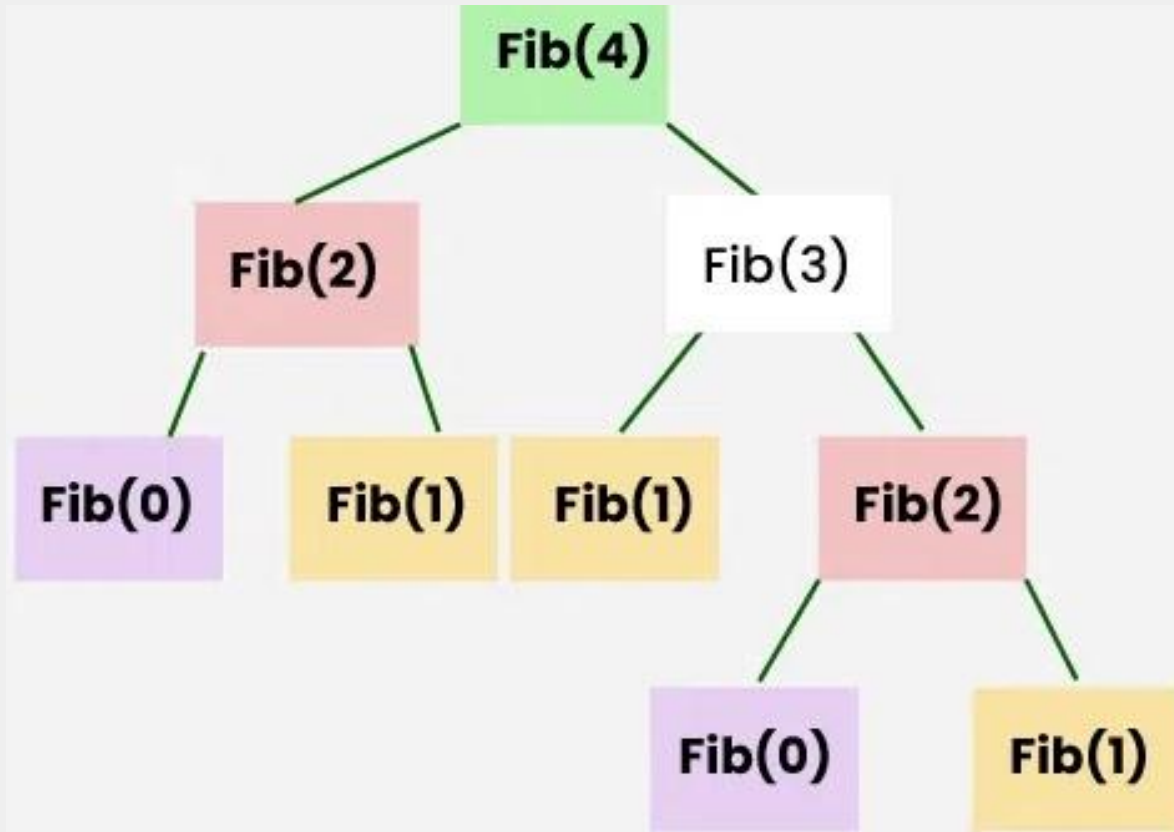
What it means is that recursion allows you to express the value of a function in terms of other values of that function.

Where the common sense tells you that if you implement your function in a way that the recursive calls are done in advance, and stored for easy access, it will make your program faster.

This is what we call Memoization - it is memorizing the results of some specific states, which can then be later accessed to solve other sub-problems.



## Example - Fibonacci



# Example - Fibonacci

**A code for it using pure recursion:**

```
int fib (int n) {  
    if (n < 2)  
        return 1;  
    return fib(n-1) + fib(n-2);  
}
```

**Using Dynamic Programming approach with memoization:**

```
void fib () {  
    fibresult[0] = 1;  
    fibresult[1] = 1;  
    for (int i = 2; i < n; i++)  
        fibresult[i] = fibresult[i-1] + fibresult[i-2];  
}
```

In the recursive code, a lot of values are being recalculated multiple times. We could do good with calculating each unique quantity only once.

# Types of DP

Majority of the Dynamic Programming problems can be categorized into two types:

1. Optimization problems.
2. Combinatorial problems.

The optimization problems expect you to select a feasible solution, so that the value of the required function is minimized or maximized.

Combinatorial problems expect you to figure out the number of ways to do something, or the probability of some event happening.

# Schema

Every Dynamic Programming problem has a schema to be followed:

- Show that the problem can be broken down into optimal sub-problems.
- Recursively define the value of the solution by expressing it in terms of optimal solutions for smaller sub-problems.
- Compute the value of the optimal solution in bottom-up fashion.
- Construct an optimal solution from the computed information.

# Bottom up vs. Top Down

- Bottom Up - I'm going to learn programming. Then, I will start practicing. Then, I will start taking part in contests. Then, I'll practice even more and try to improve. After working hard like crazy, I'll be an amazing coder.
- Top Down - I will be an amazing coder. How? I will work hard like crazy. How? I'll practice more and try to improve. How? I'll start taking part in contests. Then? I'll practicing. How? I'm going to learn programming.

In Top Down, you start building the big solution right away by explaining how you build it from smaller solutions. In Bottom Up, you start with the small solutions and then build up.

One can think of dynamic programming as a table-filling algorithm: you know the calculations you have to do, so you pick the best order to do them in and ignore the ones you don't have to fill in.

# Bottom up vs. Top Down

Let's look at a sample problem:

Let us say that you are given a number  $N$ , you've to find the number of different ways to write it as the sum of 1, 3 and 4.

For example, if  $N = 5$ , the answer would be 6.

$$1 + 1 + 1 + 1 + 1$$

$$1 + 4$$

$$4 + 1$$

$$1 + 1 + 3$$

$$1 + 3 + 1$$

$$3 + 1 + 1$$

# Bottom up

**Sub-problem:**  $DP_n$  be the number of ways to write  $N$  as the sum of 1, 3, and 4.

**Finding recurrence:** Consider one possible solution,  $n = x_1 + x_2 + \dots + x_n$ . If the last number is 1, the sum of the remaining numbers should be  $n - 1$ . So, number of sums that end with 1 is equal to  $DP_{n-1}$ . Take other cases into account where the last number is 3 and 4. The final recurrence would be:

$$DP_n = DP_{n-1} + DP_{n-3} + DP_{n-4}.$$

Take care of the base cases.  $DP_0 = DP_1 = DP_2 = 1$ , and  $DP_3 = 2$ .

Implementation:

```
DP[0] = DP[1] = DP[2] = 1; DP[3] = 2;
for (i = 4; i <= n; i++) {
    DP[i] = DP[i-1] + DP[i-3] + DP[i-4];
}
```

# Core Idea

In programming, Dynamic Programming is a powerful technique that allows one to solve different types of problems in time  $O(n^2)$  or  $O(n^3)$  for which a naive approach would take exponential time.

The intuition behind dynamic programming is that we trade space for time, i.e. to say that instead of calculating all the states taking a lot of time but no space, we take up space to store the results of all the sub-problems to save time later.



# Challenge!

"Imagine you have a collection of  $N$  wines placed next to each other on a shelf. For simplicity, let's number the wines from left to right as they are standing on the shelf with integers from 1 to  $N$ , respectively. The price of the  $i$ th wine is  $p_i$ . (prices of different wines can be different).

Because the wines get better every year, supposing today is the year 1, on year  $y$  the price of the  $i^{\text{th}}$  wine will be  $y \cdot p_i$ , i.e.  $y$ -times the value that current year.

You want to sell all the wines you have, but you want to sell exactly one wine per year, starting on this year. One more constraint - on each year you are allowed to sell only either the leftmost or the rightmost wine on the shelf and you are not allowed to reorder the wines on the shelf (i.e. they must stay in the same order as they are in the beginning).

You want to find out, what is the maximum profit you can get, if you sell the wines in optimal order?"



**Questions?**

# More Challenges

[https://cp-algorithms.com/dynamic\\_programming/intro-to-dp.html](https://cp-algorithms.com/dynamic_programming/intro-to-dp.html)