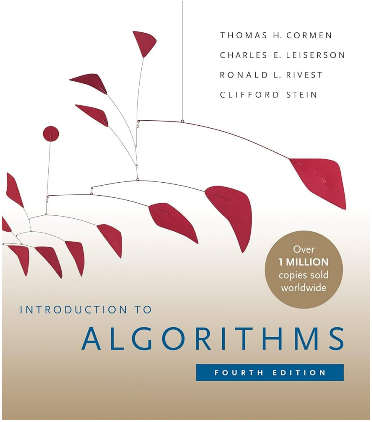
Introduction to Algorithms
Lecture 5: Dynamic Programming (DP)

Prof. Charles E. Leiserson and Prof. Erik Demaine
Massachusetts Institute of Technology

March 11, 2025

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at `https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.

Plan

Dynamic Programming

Dynamic Programming[1] (DP)

- Invented by Richard Bellman in 1950s.
- Desing technique, like Divide & Conquer.
- Applies when the subproblems overlap –that is, when subproblems share subsubproblems.
- It solves each subsubproblem just once and then saves its answer in a table.
- DP typically applies to optimization problems:
  - have many possible solutions.
  - find a solution with the optimal (min or max) value.
  - *an* optimal solution, not *the* optimal solution.

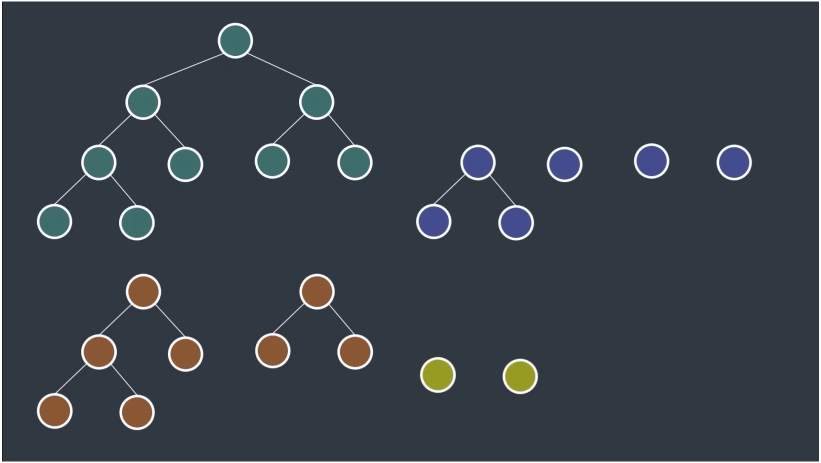
[1] *Programming* in this context refers to a tabular method, not to writing computer code.

DP notions

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution based on optimal solutions of subproblems.
3. Compute the value of an optimal solution in bottom-up fashion (recursion & memoization).
4. Construct an optimal solution from the computed information.

DP = **Recursion** + **Memoization**

DP notions



N-th Fibonacci Number[2]

Write a function that returns the n-th Fibonacci number.

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

n	1	2	3	4	5	6	7
F_n	1	1	2	3	5	8	13

[2] *Mastering Dynamic Programming - How to solve any interview problem (Part 1)*. Tech With Nikola Channel, 2024. YouTube, available at `https://youtu.be/Hdr64lKQ3e4?si=ycTe-hoyfaICRWXt`

Naive Approach

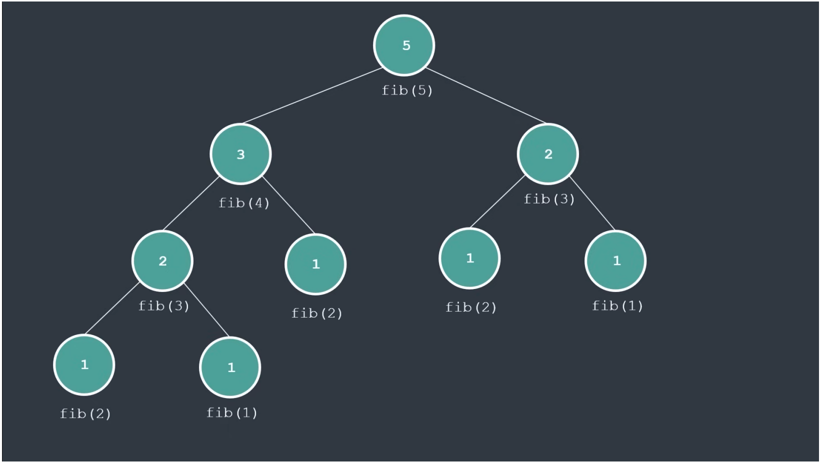
```python
def fib(n):
    if n <= 2:
        result = 1
    else:
        result = fib(n - 1) + fib(n - 2)
    return result
```

Naive Approach

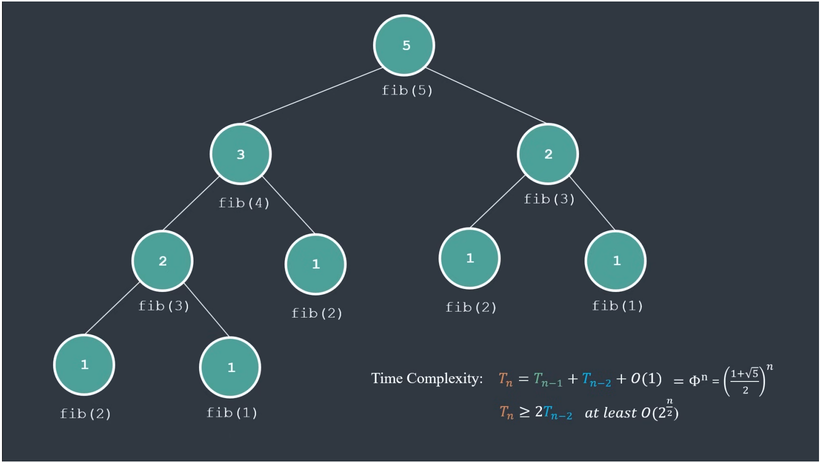
```python
def fib(n):
    if n <= 2:
        result = 1
    else:
        result = fib(n - 1) + fib(n - 2)
    return result
```

```
print(fib(7))
Output: 13
print(fib(50))
Output: ???
```

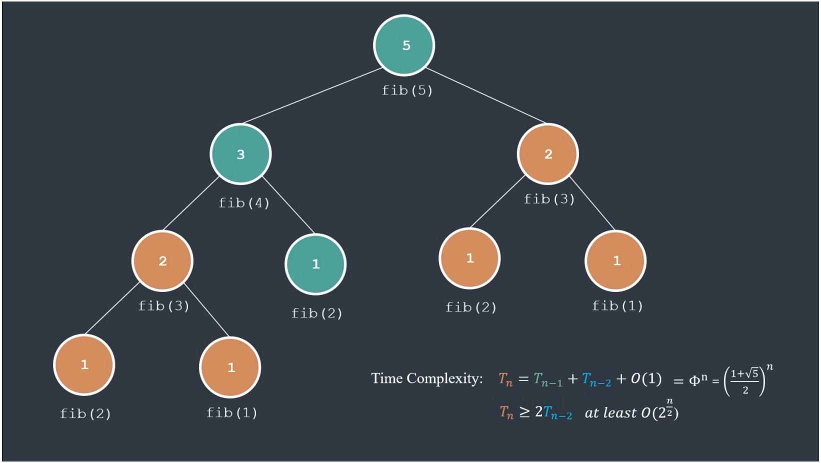
Naive Approach



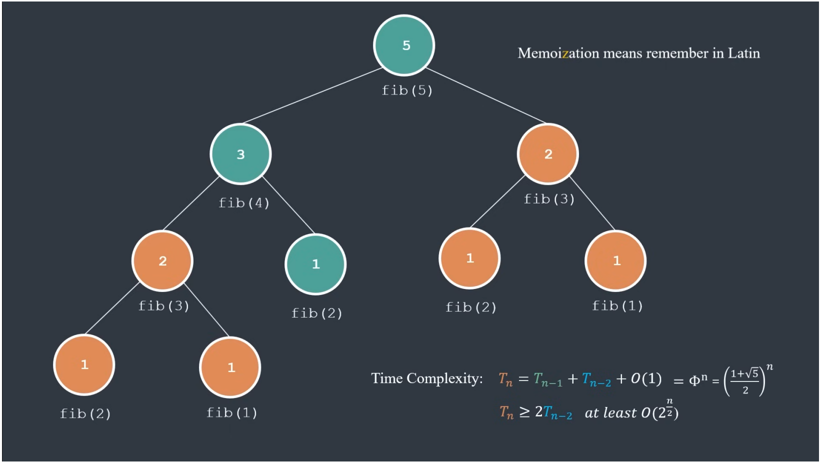
Naive Approach



Naive Approach



Naive Approach



Memoization Approach

```python
memo = {} # adding a dictionary...
def fib(n):
    if n in memo: # asking if n is in dict...
        return memo[n] # if so, we are done!
    if n <= 2:
        result = 1
    else:
        result = fib(n - 1) + fib(n - 2)
    return result
```

Memoization Approach

```python
memo = {} # adding a dictionary...
def fib(n):
    if n in memo: # asking if n is in dict...
        return memo[n] # if so, we are done!
    if n <= 2:
        result = 1
    else:
        result = fib(n - 1) + fib(n - 2)
    return result
```

```
print(fib(7))
Output: 13
print(fib(50))
Output: 12586269025
```

Memoization Approach

```python
memo = {} # adding a dictionary...
def fib(n):
    if n in memo: # asking if n is in dict...
        return memo[n] # if so, we are done!
    if n <= 2:
        result = 1
    else:
        result = fib(n - 1) + fib(n - 2)
    return result
```

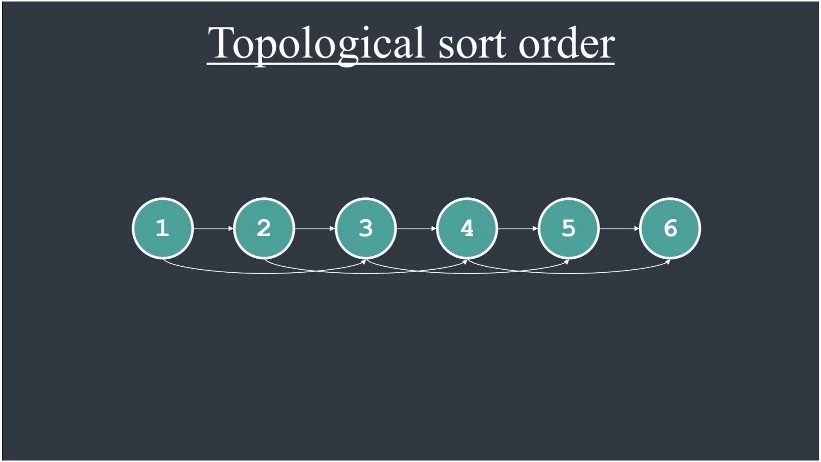
```
print(fib(7))
Output: 13
print(fib(50))
Output: 12586269025
```

Time Complexity: $O(n)$.

DP = Recursion + Memoization

Bottom-up Approach

```python
def fib(n):
    memo = {}
    for i in range(1, n + 1):
        if i <= 2:
            result = 1
        else:
            result = memo[n - 1] + memo[n - 2]
        memo[i] = result
    return memo[n]
```


Topological sort order
1
2
3
4
5
6

Longest Palindromic Sequence

Definition:

A palindrome is a string that is unchanged when reversed.

- Examples: `radar`, `civic`, `t`, `bb`, `redder`.
- Given: A string $X[1 \ldots n]$, $n \geq 1$.
- To find: Longest palindrome that is a subsequence.
- Example: Given "c h a r a c t e r".
- Output: "c a r a c".
- Answer will be ≥ 1 in length.

Strategy

$L(i, j)$: length of longest palindromic subsequence of $X[i \dots j]$ for $i \leq j$.

```
def L(i, j):
    if i == j:
        return 1
    if X[i] == X[j]:
        if i + 1 == j:
            return 2
        else:
            return 2 + L(i + i, j - 1)
    else:
        return max( L(i + 1, j), L(i, j - 1) )
```

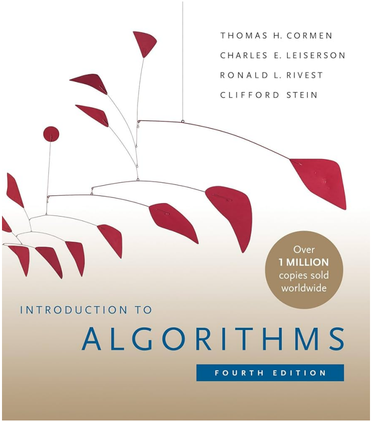
Analysis

As written, program can run in exponential time: suppose all symbols $X[i]$ are distinct.

$$
\begin{aligned}
T(n) &= \text{running time on input of length } n \\
T(n) &= \begin{cases} 1 & n = 1 \\ 2T(n-1) & n > 1 \end{cases} \\
&= 2^{n-1}
\end{aligned}
$$

End of Lecture 5.

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at
`https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.