UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

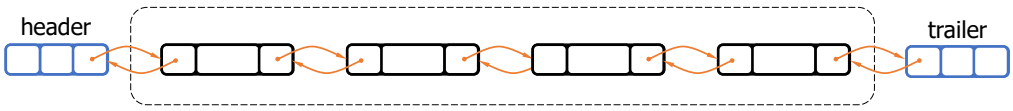
(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

Quick Recap

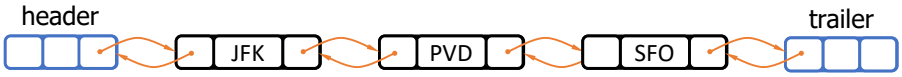
- Stacks, Queues, Linked Lists
- Doubly linked list



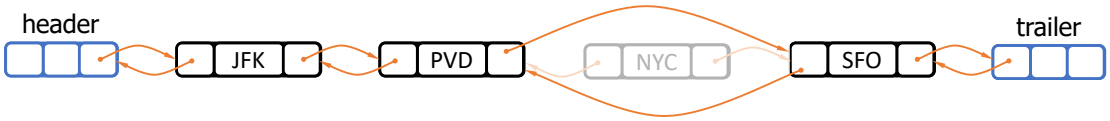
- Advantages: All operations are uniform → makes the implementation easier
- Operations can be executed in O(1)

Quick Recap

- Insertion of a node



1. Create a new node with reference of `predecessor` and `successor` nodes

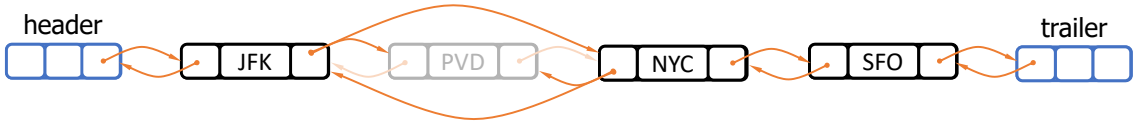


2. Then update _next and _prev pointers of predecessor and successor nodes to point to the new node

Quick Recap

- Deletion of a node

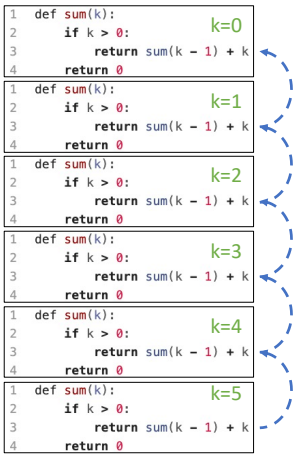
1. Link the neighbors of the node (to be deleted) with each other



Quick Recap

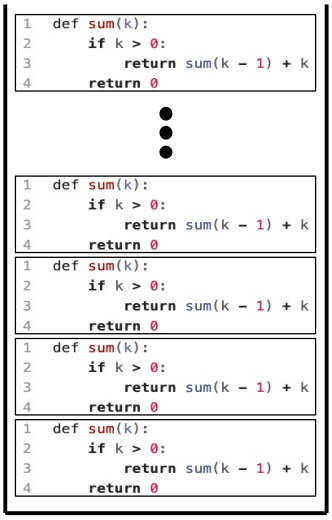
- Recursion
  → Solving a problem using function that recursively call itself.

```
1  def sum(k):
2      if k > 0:
3          return sum(k - 1) + k
4      return 0
5  print(sum(5))
```



Quick Recap

- FunctionCall Stack
  - Used Stack (LIFO) structure to hold recursive calls
  - In Pythong, infinite recursive calls are restricted to 1000 (by default)
    - Can be modified using `sys.setrecursionlimit(N)`



FunctionCall Stack

Recursion Basic Rules

1. Have a Base Case:

```
1  def sum(k):
2      if k > 0:
3          return sum(k - 1) + k
4      return 0
```

2. Recursion calls should move towards the base case

Activity

Write an algorithm to calculate the factorial of a number.
E.g. 4! ➔ 4x3x2x1 = 24

For Anonymous Questions

Activity Solution

Write an algorithm to calculate the factorial of a number.
E.g. 4! ➔ 4x3x2x1 = 24

```
def fact_it(k):
    fact = 1
    for i in range(k, 1, -1):
        fact *= i
    return fact
```

```
def fact_recr(k):
    if k >= 1:
        return k * fact_recr(k-1)
    else:
        return 1
```


Recursion Fibonacci Sequence

- Sequence is named after Leonardo Fibonacci
- Fibonacci numbers exist everywhere, from petals in a flower to bones in fingers, etc.

It is series of numbers in which a given number is a sum of previous two numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, …..

Mathematically, it is represented as:

$$f(n) = f(n - 1) + f(n - 2)$$

Recursion | Fibonacci Sequence

Golden Ratio

$$\phi = \frac{A+B}{A} = \frac{A}{B}, \text{ where A > B}$$

Golden ratio of 1.6 exist between two Fibonacci numbers

Recursion | Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, …..

$$fib(k) = \begin{cases} 0 & if\ k = 0 \\ 1 & if\ k = 1 \\ fib(k-2) + fib(k-1) & if\ k > 1 \end{cases}$$

Activity

0, 1, 1, 2, 3, 5, 8, 13, 21, …..

$$fib(k) = \begin{cases} 0 & if\ k = 0 \\ 1 & if\ k = 1 \\ fib(k\ -2) + fib(k-1) & if\ k > 1 \end{cases}$$

(a) Write a recursive algorithm to calculate the next number in the Fibonacci sequence.
(b) determine how many recursive function calls will be made for k = 5.

CSE-2050 – Data Structures and Object-Oriented Design
Recursion and Dynamic Programming

Activity

0, 1, 1, 2, 3, 5, 8, 13, 21, …..

$$fib(k) = \begin{cases} 0 & if\ k = 0 \\ 1 & if\ k = 1 \\ fib(k\ -2) + fib(k-1) & if\ k > 1 \end{cases}$$

(a) Write a recursive algorithm to calculate the next number in the Fibonacci sequence.
(b) determine how many recursive function calls will be made for k = 5.

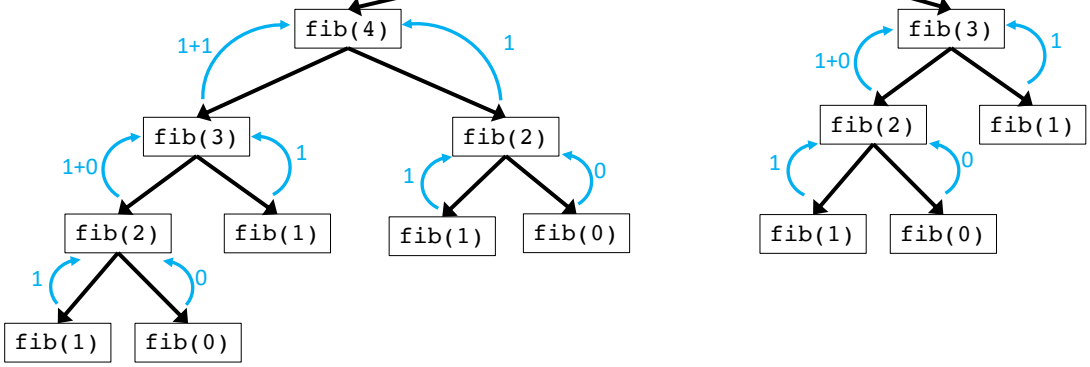
```
1  def fib(k):
2      if k in [0,1]:
3          return k
4      return fib(k-1) + fib(k-2)
```

Week 6 – 10/03 – 10/07 – Lecture 2
Hasan Baig

Activity | Visual Analysis

fib(5) → 5

```
1  def fib(k):
2      if k in [0,1]:
3          return k
4      return fib(k-1) + fib(k-2)
```



Hasan Baig

CSE-2050 – Data Structures and Object-Oriented Design
Recursion and Dynamic Programming

Recursion | Fibonacci Sequence
Timing Complexity

$$fib(k\ -2) + fib(k-1)$$

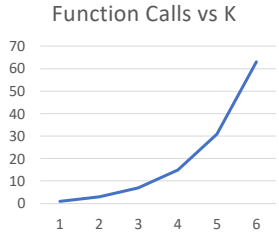
Let, T(k) denotes the number of calls/operations required to compute fib(k), then,

T(k) = T(k-1) + T(k -2) + 1

For simplicity, consider T(k-1) = T(k-2)

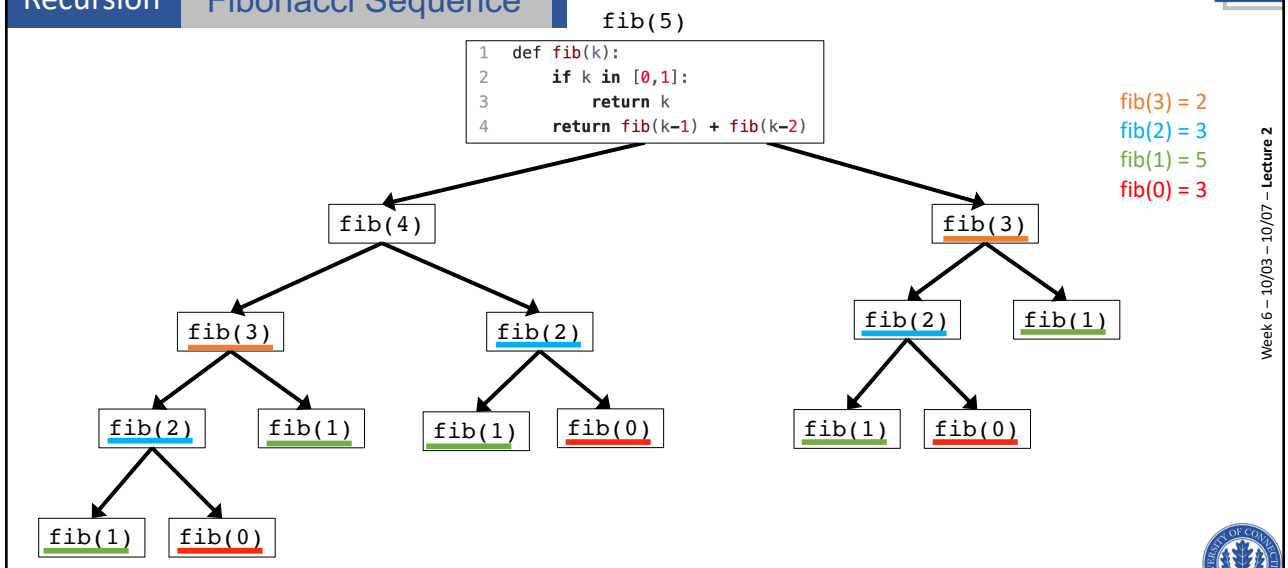
T(k) = 2T(k-1) + 1

For k = 1, → T(1) = 2T(0) + 1 → T(1) = 1
For k = 2, → T(2) = 2T(1) + 1 → 2(1) + 1 → T(2) = 3
For k = 3, → T(3) = 2T(2) + 1 → 2(3) + 1 → T(3) = 7
For k = 4, → T(4) = 2T(3) + 1 → 2(7) + 1 → T(4) = 15
For k = 5, → T(5) = 2T(4) + 1 → 2(15) + 1 → T(5) = 31
For k = 6, → T(6) = 2T(5) + 1 → 2(31) + 1 → T(6) = 63



Hasan Baig

Recursion | Fibonacci Sequence



Dynamic Programming

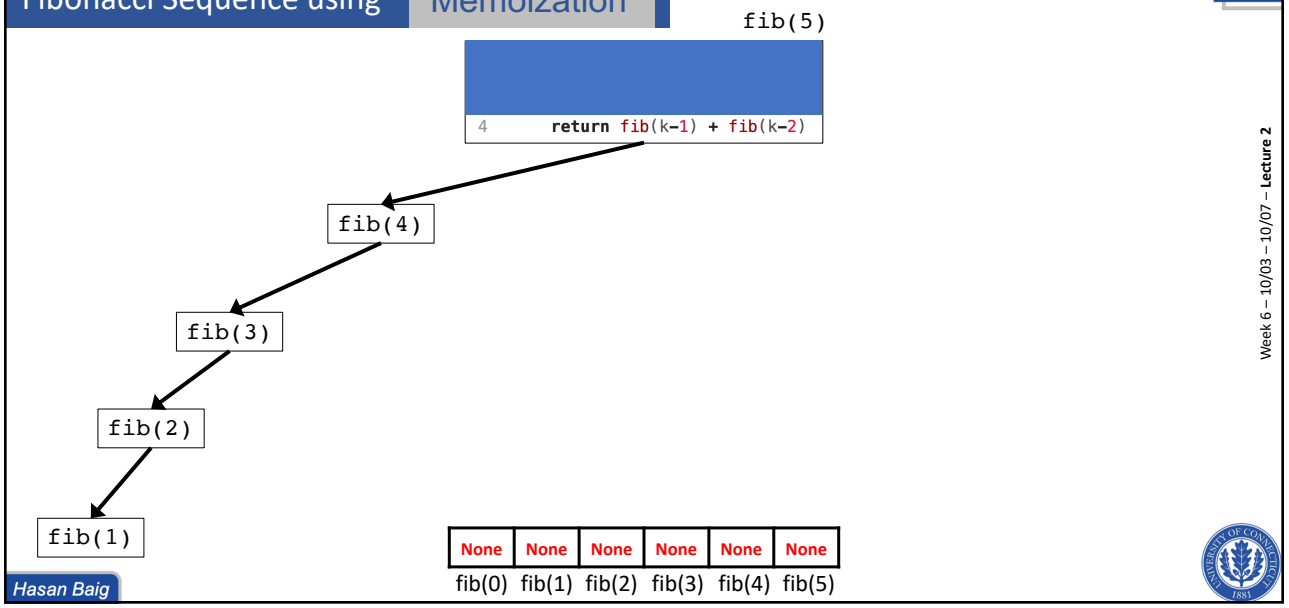
It refers to an optimization over plain recursion:

- Avoid making a function call again which has already been executed by:
  - storing the intermediate solution of subproblems and use them later wherever needed
  - This is called **Memoization**

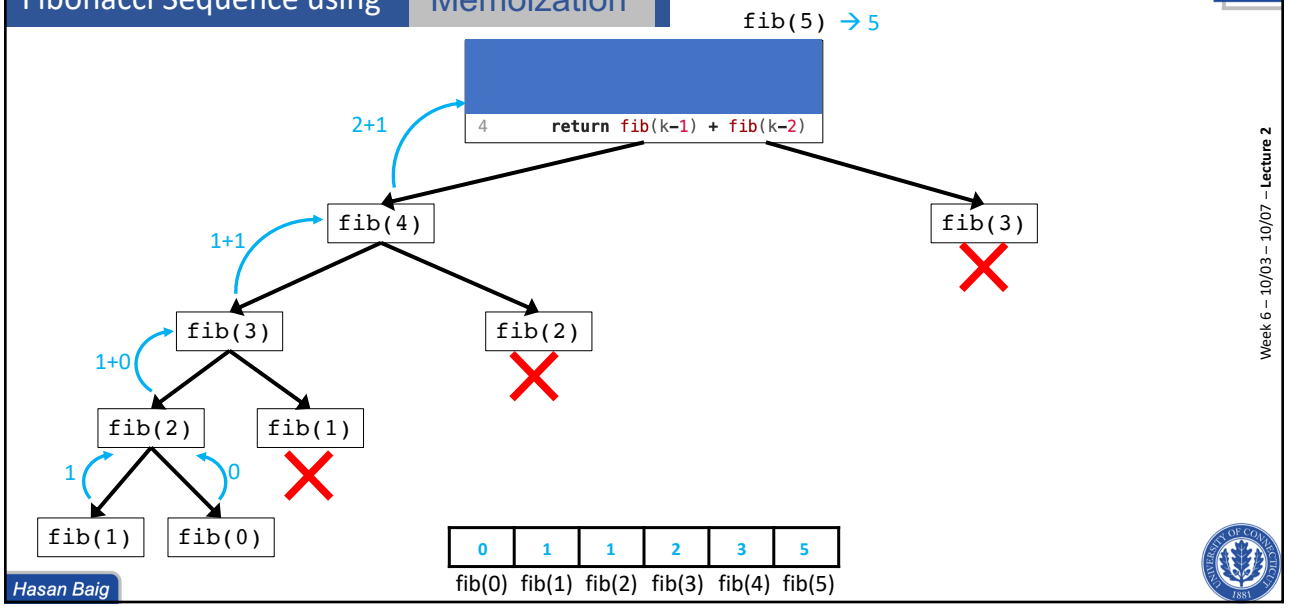
Two approaches of formulating a dynamic programming solution:

1. Top-down approach: Uses ***memoization*** technique (recursion + caching)
2. Bottom-up approach: Uses ***tabulation*** technique (solve iteratively) → Dynamic Prog

Fibonacci Sequence using Memoization



Fibonacci Sequence using Memoization

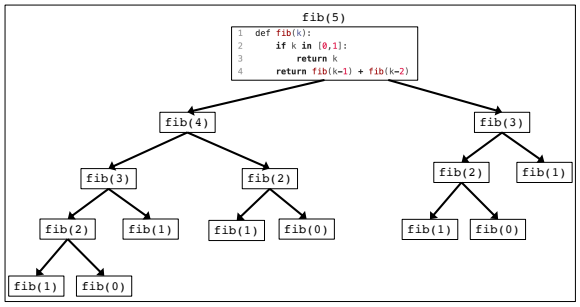


Fibonacci Sequence using Memoization

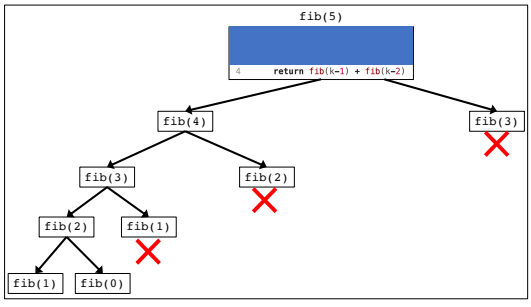
```
1  def fib_memo(k, fib_array):
2      if k in [0,1]:
3          fib_array[k] = k
4          return fib_array[k]
5      if fib_array[k] != None:
6          return fib_array[k]
7      fib_array[k] = fib_memo(k-1, fib_array) + fib_memo(k-2, fib_array)
8      return fib_array[k]
9
10 k = int(input())
11 fib_array = [None]*(k+1)
12 fib_memo(k, fib_array)
```


Fibonacci Sequence using Memoization

Total number of function calls



15



9

fib(n) = n + 3

O(n)

The complexity reduces from exponential to polynomial or linear