CSC373

ALGORITHMS

DYNAMIC PROGRAMMING

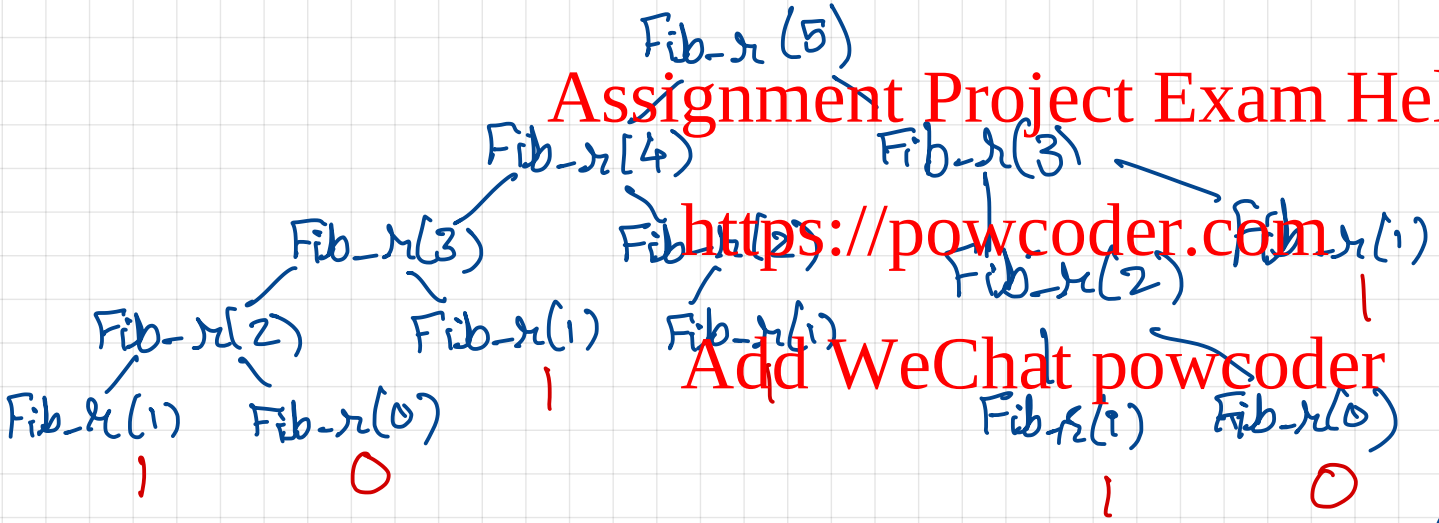
FIBONACCI NUMBERS

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

$FIB(0) = 0$

$FIB(1) = 1$

```
def FIB_r(n):
    if n==0:
        RETURN 0
    if n==1:
        RETURN 1
    RETURN (FIB_r(n-1)
            + FIB_r(n-2))
```

Claim: Correct

Pf: Induction

Claim2: Time taken by $Fib\_r(n) \geq Fib(n)$

Claim3: $Fib(n) \geq 2^{n/2} - 1$

Increasing EXPONENTIALLY

$\Omega(2^{n/2})$ time!

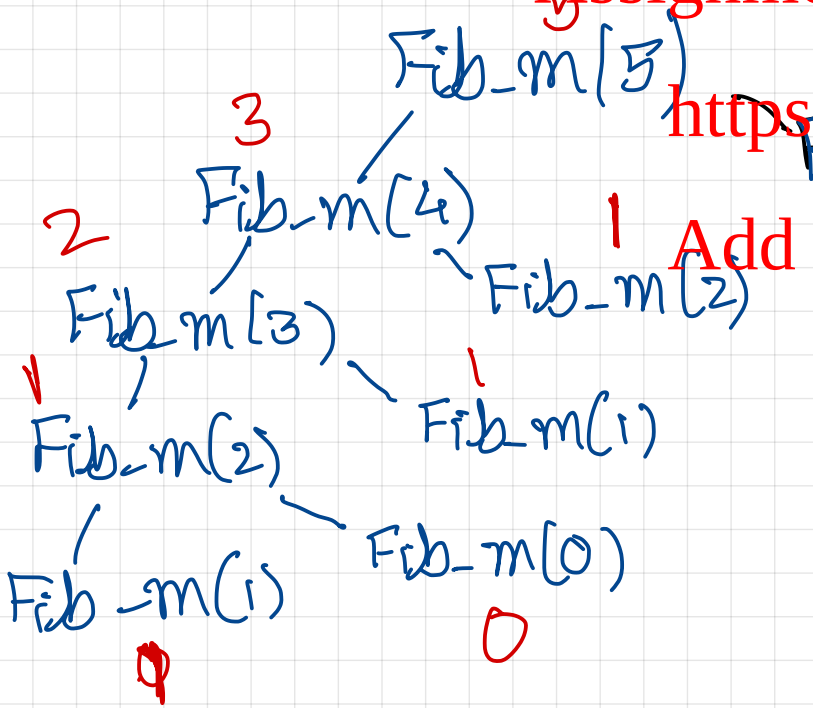
KEY IDEA: AVOID REPETITIVE COMPUTATIONS BY STORING RESULTS

Fib_array

0	1	2	3	4	5
0	1	1	2	3	5
0	1	-1	-1	-1	-1

```
fib_array = 50 * [-1]
fib_array[0] = 0
fib_array[1] = 1
def fib_m(n):
    if fib_array[n] == -1
        fib_array[n] =
            fib_m(n-1)
          + fib_m(n-2)
    return fib_array[n]
```

MEMOIZATION



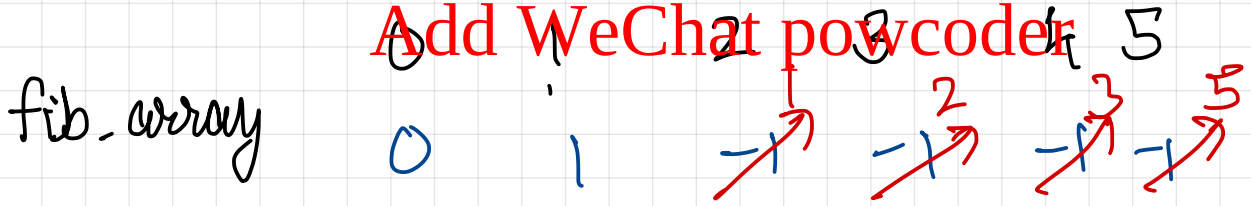
Claim: Every fibonacci number is computed exactly once

TOTAL RUNNING TIME = $O(n)$

OBSERVATION:

NO NEED TO RECURSE

FILL UP ARRAY BOTTOM-UP




```
def fib_nr(n):
    fib_array = [0] * [n]
    fib_array [0] = 0
    fib_array [1] = 1
    for i in range (2 : n+1):
        fib_array [i] = fib_array [i-1]
                      + fib_array [i-2]
```


Running time : $O(n)$

RECURSIVE
- REPITITION
- MEMOIZATION
- NON-RECURSIVELY BOTTOM-UP

DYNAMIC PROGRAMMING

MAX-WEIGHTED INTERVAL SCHEDULING

Given: n intervals s_i : starting

f_i : ending

w_i : weight

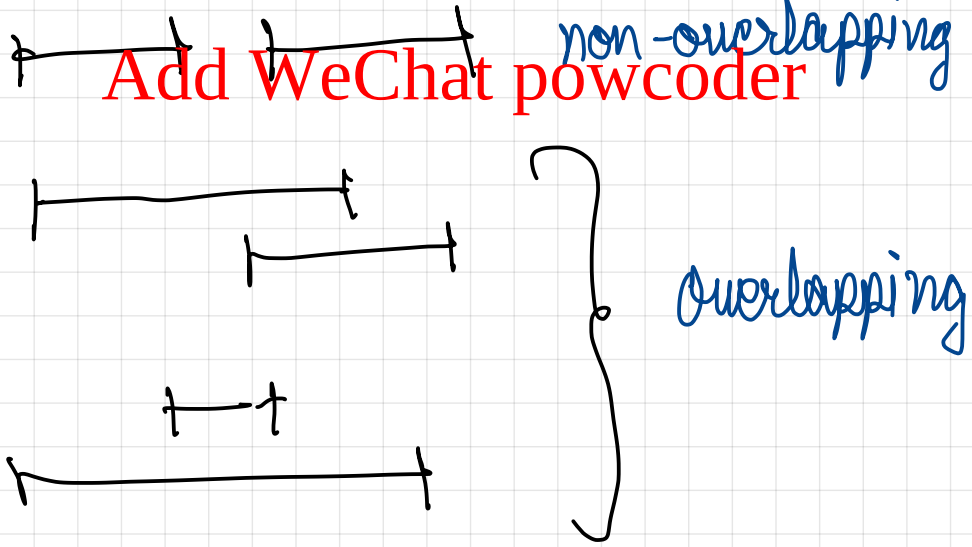
Goal:

Find a collection I of intervals

that is non-overlapping & has maximum weight

$\sum_{i \in I} w_i$

OR $f_i < s_j$ or $f_j < s_i$



Q1 : Is there any structure to the overlap?

Can easily determine all intervals that don't overlap with a fixed interval j

want $f_i < s_j$

} Note non-overlapping on one side

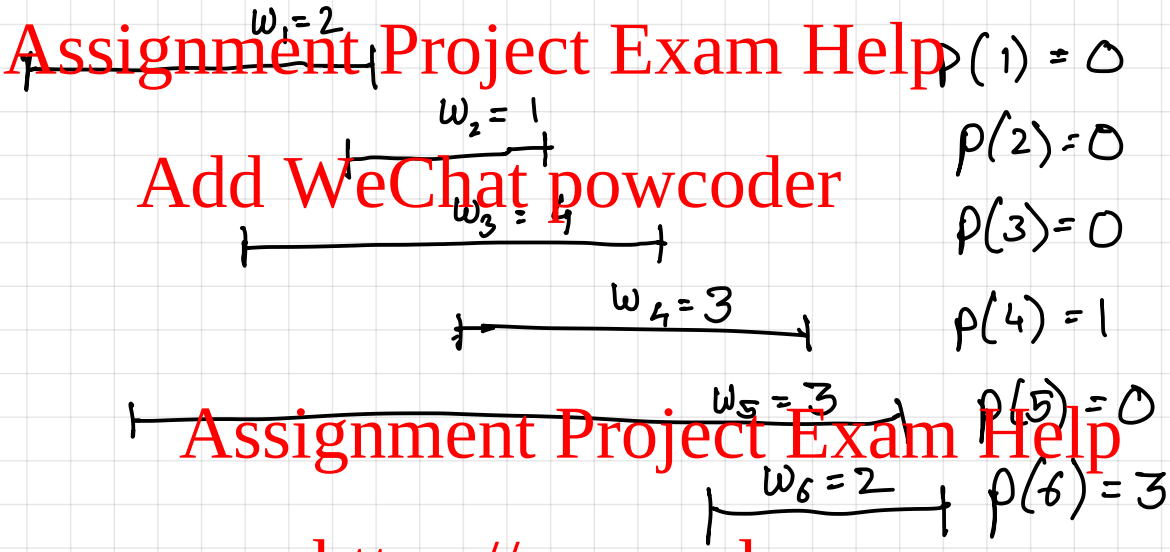
Idea: Sort all intervals by their finishing time

Assume : $f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_n$

$p[j]$: largest index $i < j$ such that $f_i < s_j$ $\Leftrightarrow$ largest i non-overlapping with j

Computed using binary search

MAXIMUM WEIGHT INTERVAL SCHEDULING



All intervals $\{1, \ldots, p[j]\}$ dont overlap with j

All " $\{p[j]+1, \ldots, j-1\}$ overlap w/ j

$OPT[\{1, \ldots, j\}]$

Either j is not in best soln

j is in the best soln

assumption

$OPT(\{1, \ldots, j-1\})$ Claim $OPT = \{j\} \cup OPT(\{1, 2, \ldots, p[j]\})$

$$OPT(\{1, 2, \ldots, j\}) = \begin{cases} 0 & j = 0 \\ \max(OPT\{1, \ldots, j-1\}, \\ \quad w_j + OPT(\{1, 2, \ldots, p[j]\}) \end{cases}$$