

# Data structure and Algorithms

## Course

Created by : Israa Abdelghany  
LinkedIn : [www.linkedin.com/in/israa-abdelghany-4872b0222](https://www.linkedin.com/in/israa-abdelghany-4872b0222)  
GitHub : <https://github.com/IsraaAbdelghany9>

## Review Last Lecture

### → time Complexity :-

- concerns about the relation between # of inputs with time
- relation between  $n$  & number of iterations (time)

Example :-

for ( $i=1$  ;  $i < n$  ;  $i = i^2$ )

$\Sigma$   
 statement;  
 $\Sigma$

$n$	no. of iter.
$2^1$	1
$2^2$	2
$2^3$	3
$2^4$	4

$\therefore \log_2 n$

for ( $i=1$  ;  $i^*i < n$  ;  $i++$ )

$\Sigma$   
 statement ;  
 $\Sigma$

$n$	iter.
4	1
8	2
16	3

$$\log(n) - 1 = \log_2 n$$

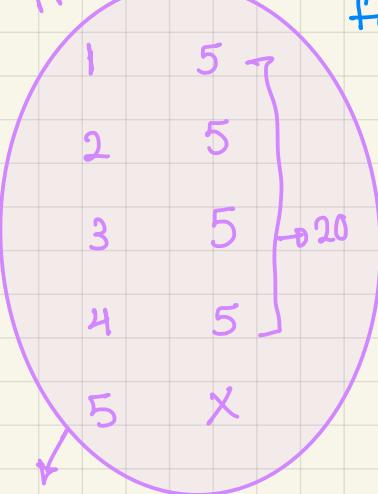
for ( $i=1$  ;  $i^*i < n$  ;  $i = i^2$ )

$\Sigma$   
 statement ;  
 $\Sigma$

$n$	iter.
2	1
4	1
8	1, 1
16	1, 1
32	1, 1, 1
64	1, 1, 1

$$\log_2(n)$$

if i starts with 1



for ( $i = 0 ; i < n ; i++$ )

for ( $j = 0 ; j < n ; j++$ )

statement

i	iter.
1	5
2	5
3	5
4	5
5	5
6	X

$O(n^2)$  also

for ( $i = 0 ; i < n ; i++$ )

{

for ( $j = 0 ; j < i ; j++$ )

{

statement

}

time is a series.

$$t = \frac{n * (n+1)}{2}$$

$$= \frac{n^2 + n}{2}$$

$O(n^2)$

assume  $n = 5$

i	iteration
0	0
1	1
2	2
3	3
4	4
5	X

for (i=1; i<n; i\*=2)

{

    for (j=0; j<i; j++)

{

        statement

}

}

$n=8$

i	iter
1	1
2	2
4	4

$$\begin{aligned} & 1 + 2 + 4 + 8 + \dots \\ & \underbrace{1}_{2^0} \quad \underbrace{2}_{2^1} \quad \underbrace{4}_{2^2} \quad \underbrace{8}_{2^3} \quad \dots \end{aligned}$$

$O(2^{\log n})$

## Recurrence relation

Recursion tree

Back substitution

**Ex1**

$$f(n) \xrightarrow{\quad} T(n)$$

if ( $n > 0$ )  
  {

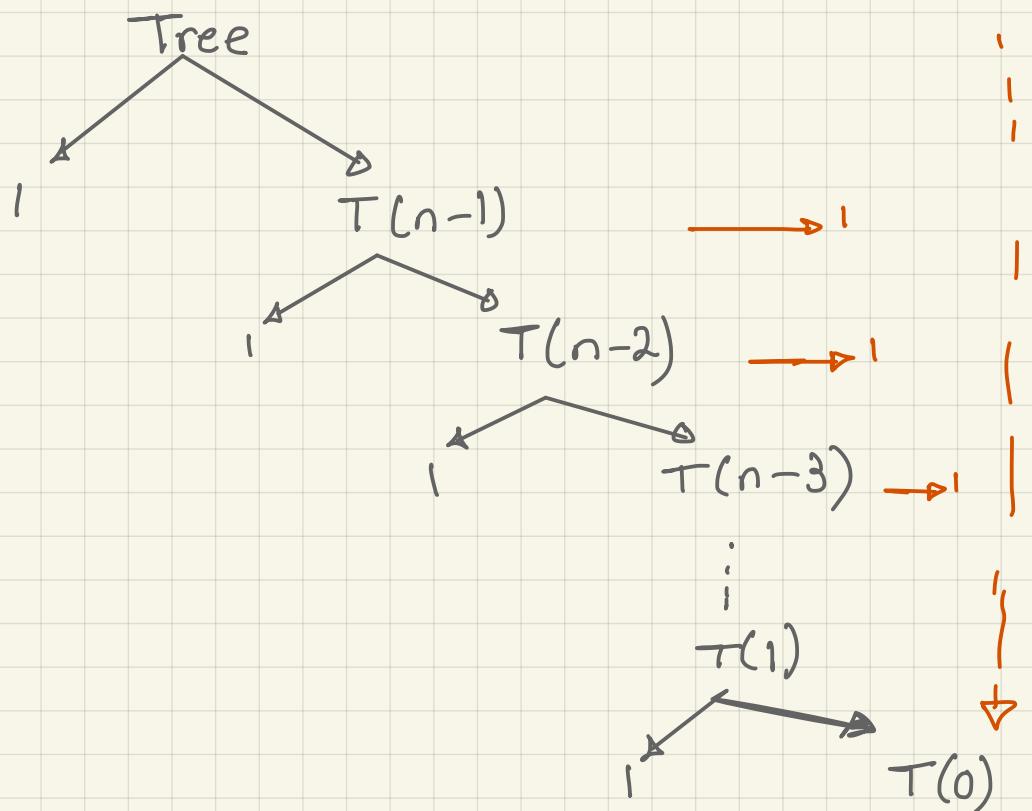
    print (n)  $\longrightarrow$  1

$f(n-1) \longrightarrow T(n-1)$

  }

Σ

$$T(n) = \begin{cases} 1 & ; n=0 \\ T(n-1)+1 & ; n>0 \end{cases}$$



$$T(n) = n + 1$$

$O(n)$

Back substitution:-

$$\text{eq } ① \leftarrow T(n) = T(n-1) + 1 \quad \xrightarrow{\hspace{1cm}} \text{I}$$
$$T(n-1) = T(n-2) + 1 \quad \xrightarrow{\hspace{1cm}} \text{II}$$
$$T(n-2) = T(n-3) + 1 \quad \xrightarrow{\hspace{1cm}} \text{III}$$
$$T(n) = (T(n-2) + 1) + 1 \quad \xrightarrow{\hspace{1cm}} \text{IV}$$
$$\text{eq } ② \leftarrow \quad \quad \quad = T(n-2) + 2$$

$$T(n) = (T(n-3) + 1) + 2$$
$$\text{eq } ③ \leftarrow \quad \quad \quad = T(n-3) + 3$$

So  $T(n) = T(n-k) + k$

$n-k=0$

$k=n$

⇒ we took into consideration  
eq (1, 2, 3) to recognize  
the pattern

$$T(n) = T(n-n) + n$$
$$= n$$

So  $O(n)$

Ex 2

f1 (int n)  $\rightarrow T(n)$

{

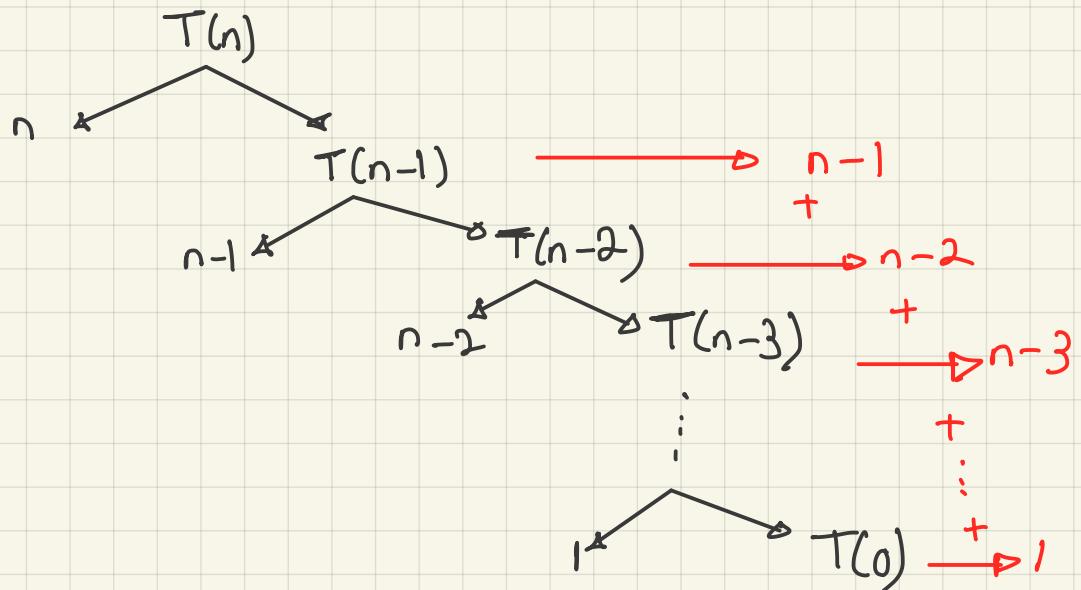
if ( $n > 0$ )

{

for (i = 0 ; i < n ; i++)  
{ print (n)  
    3 } n times

if

f1 (n-1)  $\rightarrow T(n-1)$



$$T(n) = 1 + 1 + 2 + \dots + (n-1) + n$$

$$= 1 + \frac{n*(1+n)}{2}$$

$$\Rightarrow O(n^2)$$

Back substitution:-

$$T(n) = T(n-1) + n$$

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

$$T(n-2) = T(n-3) + (n-2)$$

sub 2 in 1

$$T(n) = [ T(n-2) + (n-1) ] + n$$

$$= \underbrace{T(n-2)}_{\downarrow} + \underbrace{(n-1) + n}_{\searrow}$$

sub 3 in 4

$$T(n) = [ T(n-3) + (n-2) ] + (n-1) + n$$

$$= T(n-3) + (n-2) + (n-1) + n$$

So pattern is

$$T(n) = T(n-k) + (n - (k-1)) + (n - (k-2)) + \dots + n$$

$$n - k = 0$$

$$\text{So } n = k$$

$$T(n) = T(n-n) + (n - (n-1)) + (n - (n-2)) + \dots + n$$

$$T(n) = T(0) + 1 + 2 + \dots + n$$

Same pattern as tree

# Master theory for decreasing fun

Using Back substitution



$$T(n) = T(n-1) + 1 \quad (1)$$

$$T(n-1) = T(n-2) + 1 \quad (2)$$

$$T(n-2) = T(n-3) + 1 \quad (3)$$

(2) in (1)

$$T(n) = [T(n-2) + 1] + 1 = T(n-2) + 2 \quad (4)$$

(3) in (4)

$$T(n) = [T(n-3) + 1] + 1 = T(n-3) + 3$$

from those eqns we find that

$$T(n) = T(n-k) + k \quad \#$$

at  $T(0) \quad k=n \quad [\text{Last case}]$

$$T(n) = T(n-n) + n = T(0) + n = \boxed{n+1}$$

Same as recursion Tree  $\mathcal{O}(n)$  complexity

Master theorem for decreasing function

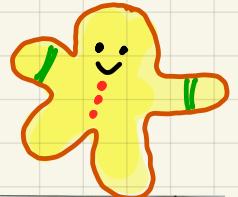
$$T(n) = aT(n-b) + f(n)$$

$$\mathcal{O}(f(n)) = \mathcal{O}(n^k) \quad a < 1$$

$$\mathcal{O}(n^k f(n)) = \mathcal{O}(n^{k+1}) \quad a = 1$$

$$\mathcal{O}(a^{n/b} * f(n)) = \mathcal{O}(a^{n/b} * n^k) \quad a > 1$$

$$\begin{cases} a, b > 0 \\ f(n) = \mathcal{O}(n^k) \end{cases}$$



- ex.  $\therefore T(n) = T(n-1) + 1 \Rightarrow \mathcal{O}(n)$
- $\therefore T(n) = T(n-1) + n \Rightarrow \mathcal{O}(n^2) = \mathcal{O}(n^2)$
- $\therefore T(n) = T(n-1) + \log n \Rightarrow \mathcal{O}(n \log n)$
- $\therefore T(n) = 2T(n-1) + 1 \Rightarrow \mathcal{O}(2^n)$
- $\therefore T(n) = 2T(n-1) + n \Rightarrow \mathcal{O}(n2^n)$
- $\therefore T(n) = 2T(n-2) + 1 \Rightarrow \mathcal{O}(2^{n/2})$



Master theory for Decreasing function

$$T(n) = aT(n-b) + f(n) \quad a > 0, b > 0 \quad \mathcal{O}(n^k) \Rightarrow k > 0$$

$$a < 1 \Rightarrow \mathcal{O}(n^k)$$

$$a = 1 \Rightarrow \mathcal{O}(n * n^k)$$

$$a > 1 \Rightarrow \mathcal{O}(a^{n/b} * n^k)$$

$$\begin{cases} T(n) = T(n-1) + 1 \quad \mathcal{O}(1) \\ T(n) = T(n-2) + n \quad \mathcal{O}(n^2) \\ T(n) = T(n-1) + \log n \quad \mathcal{O}(\log n) \end{cases}$$

$$\begin{cases} T(n) = 2T(n-1) + 1 \quad \mathcal{O}(2^n) \\ T(n) = 2T(n-2) + 1 \quad \mathcal{O}(2^m) \\ T(n) = 3T(n-3) + 1 \quad \mathcal{O}(3^n) \end{cases}$$

## Master theory for decreasing fun

Master theory for dividing		$a \geq 0$	$b > 0$	$f(n) = n^k (\log n)^p$
$T(n) = a T(n/b) + f(n)$				
* $\log_b^a$				
- Case 1: $\log_b^a > k \Rightarrow O(n^{\log_b^a})$				
- Case : $\log_b^a = k$				
$p > -1$				$O(n^k \log^{p+1} n)$
$p = -1$				$O(n^k \log \log n)$
$p < -1$				$O(n^k)$
- Case : $\log_b^a < k$				
$p \geq 0$				$O(n^k (\log n)^p)$
$p < 0$				$O(n^k)$

$$T(n) = 4 T(n/2) + n \xrightarrow{a=4, b=2} n^1 \log n^0$$

$$\log_b^a = \log_2^4 = \log_2^2 = 2 \log_2^2 = 2$$

$$O(n^2)$$

$$T(n) = 4 T(n/2) + n^2 \log n$$

$$a=4, b=2, k=2, p=1$$

$$\log_b^a = \log_2^4 = 2 = k \quad \text{case (2-1)}$$

$$O(n^2 (\log n)^2)$$

$$T(n) = 2 T(n/2) + n / \log n$$

$$a=2, b=2,$$

$$\log_b^a = \log_2^2 = 1 = k \quad \text{case (2-2)}$$

$$O(n \log \log n)$$

$$T(n) = 2 T(n/2) + n / \log n^2$$

$= k$  (case 2-3)

$$O(n)$$

$$T(n) = T(n/2) + n^2 \longrightarrow \Theta(n^2 (\log n)^0)$$

$$a=1, b=2, k=2, P=0$$

$$\log_2 1 = 0 < k \quad \text{Case 3-1}$$
$$O(n^2)$$

---

$$T(n) = 4(T/2) + n^3/\log n$$

$$a=4, b=2, k=3, P=-1$$

$$\log_b a = \log_2 4 = 2 < k \quad \text{case (3-2)}$$

$$O(n^3)$$

# Greedy Technique



- optimal local solution  $\Rightarrow$  hope it will lead to Global solution

- optimal solution seeks for
  - $\rightarrow \min$
  - or
  - $\rightarrow \max$

3 steps:-

- create candidate set :-
  - $\hookrightarrow$  all Candidates
- feasibility set :-
  - $\hookrightarrow$  define candidates meets specific requirements
- optimal solution :-
  - $\hookrightarrow$  find best solution

→ Greedy :- is about the Local best solution but overall (global) best solution may not be selected because I'm Looking on the best choice in each solution

Knapsack :- 'an application of Greedy'

types , - ① divisible  $\Rightarrow$  Greedy

② non divisible  $\Rightarrow$  dynamic Prog.

— calculate value/weight

## Job sequencing with deadline:-



- Sort projects according to profit
- $d_i \rightarrow$  deadline       $1 < r \leq d_i$        $[r-1, r]$

# "Dynamic Programming"

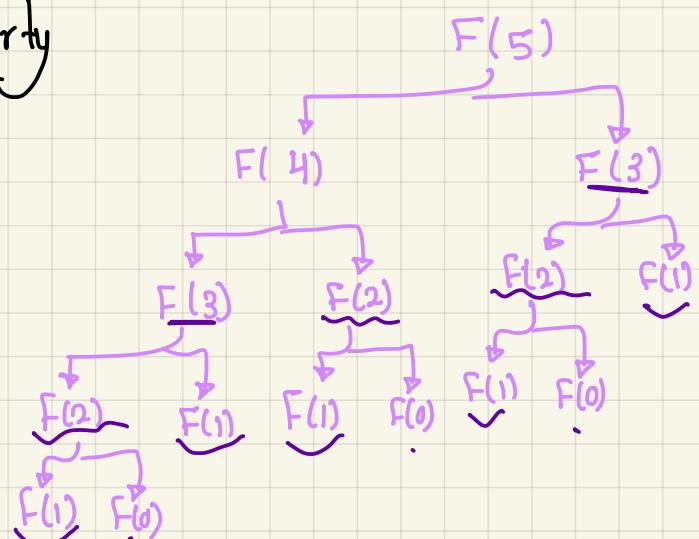
→ optimal solution

characteristics:-

- overlapping sub-problems
- optimal structure property

0, 1, 1, 2, 3, 5, 8, 13

Fibonacci  
Numer

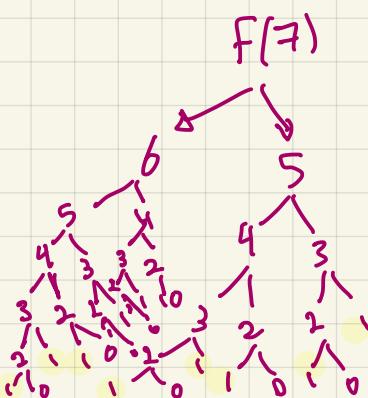


## Dynamic programming Methods:-

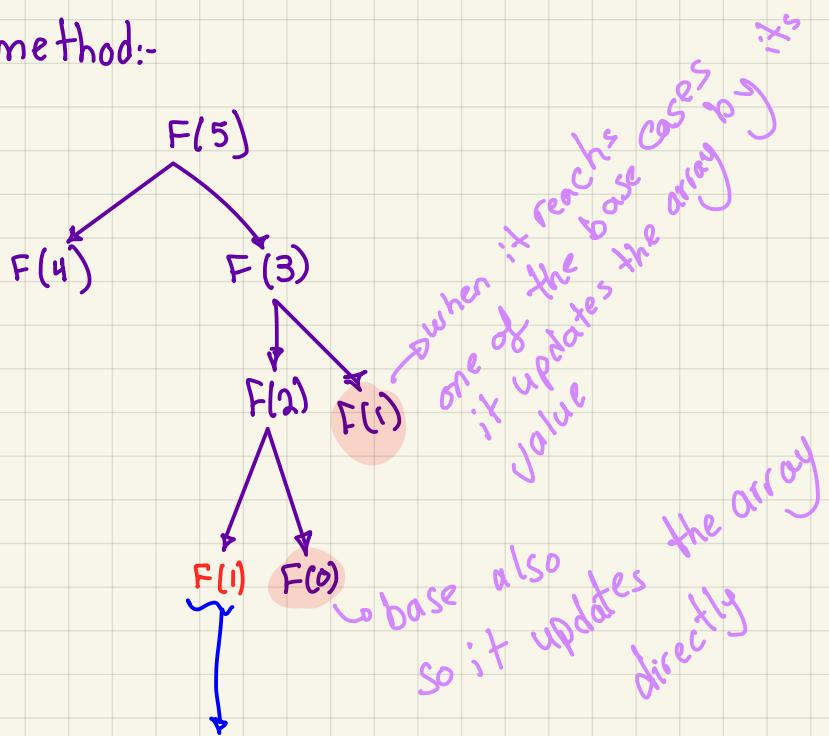
- 1 Top-down with Memoization
- 2 Bottom-up with Tabulation

$F(0) \rightarrow F(1) \rightarrow F(2) \dots$  etc

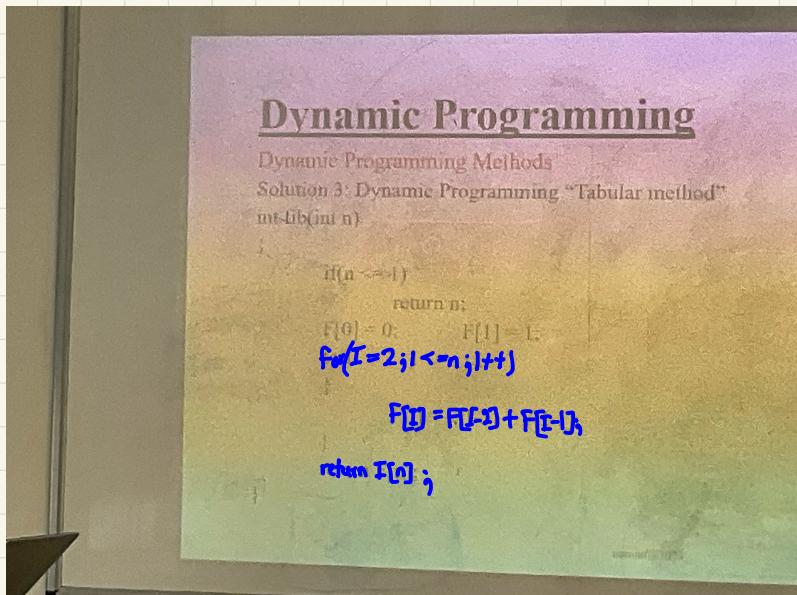
$$Fib(n) = \begin{cases} \text{Base Case} & \text{if } n \leq 1 \\ \text{Recursive Case} & \text{otherwise} \end{cases}$$



memorize method:-



if I wrote it with red that means  
I didn't call the function  $\Rightarrow$  it's already saved  
in the array

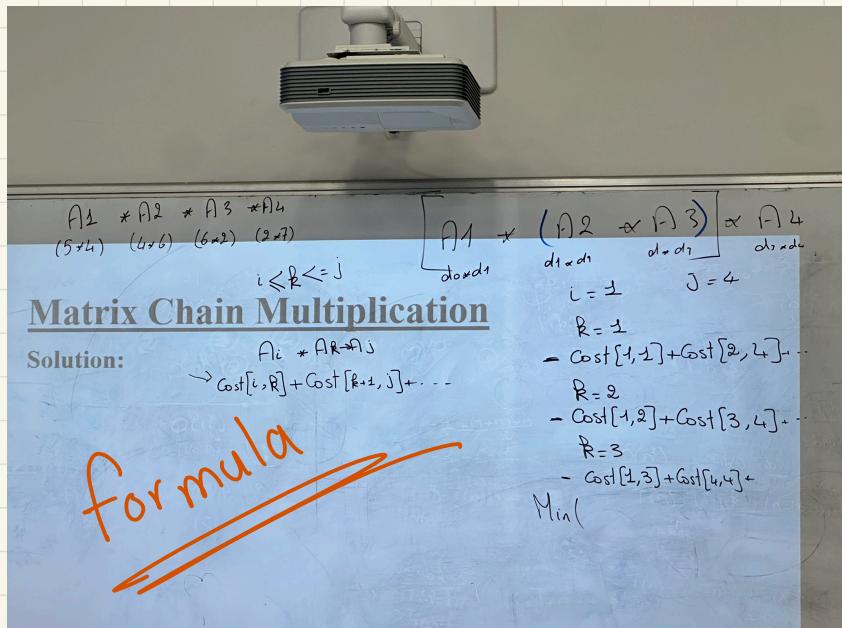


# Steps to solve Dynamic programming :-

Slides  
≡

## Matrix chain multiplication :-

- Same concept of greedy algorithm.
- see locally what's the best solution.



Non divisible

Knapsack

→ in the table I do combinations between 1, 2 and 2, 3 and finally them both (in the 3<sup>rd</sup> item)  
and so on for all other following, you should consider the cases before