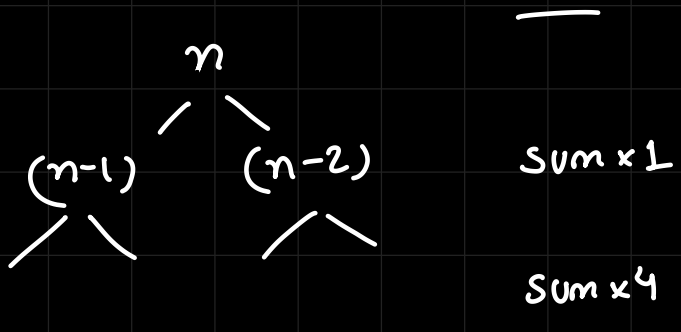
# Dynamic Programming

## ① FIBONACCI

$$F(n) = F(n-1) + F(n-2) \quad \{ F(0) = F(1) = 1 \}$$

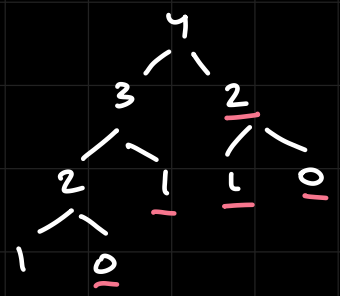
every time we want to find F(n), Fib is calculated every single time for every number less than n.




to prevent this high time complexity, we store the child values in an array so that it is not computed multiple times in one run

In divide and conquer, almost every subproblem is independent and hence DP is not useful there.

DP is very useful if subproblems are connected




even at small n we get dupes for n >>, DP is very useful

memoization

# Non Adjacent Heavy Set

Given: $n$ balls (weighted)
create a sequence of non adjacent balls with max weight

Trial 1: pick ball with highest weight s.t. it is not adjacent

eg: (1) (5) (8) (5) (4) (7)

↳ 16

but the optimized set is (5) (5) (7)
↳ 17

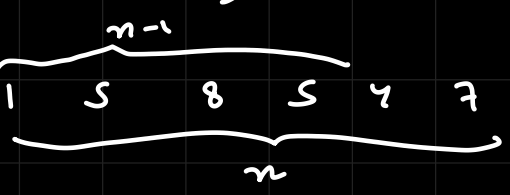
Subproblem:

MaxWt[n] = weight of an optimal set
$B^* \subset \{b_1, b_2, b_3 \ldots b_k\}$
s.t. no 2 balls are adj and sum is maximized

if 7 is not in B* lets say,
then MaxWt is already maximized till 7

if 7 is not in optimal set

$$MaxWt(n) = MaxWt(n-1)$$




If 7 is in B*,

4 must be not a part of B*
(neighbour)

also, $maxwt(n) = maxwt(n-2) + 7$

↓
not (n-1) because the neighbour of 7 wont be in B*

Recurrence:

for $k > 2$:

$$maxWt[k] = max\{ MaxWt(n-2) + b_k, MaxWt(k-1) \}$$

CLAIM: if $b_k \in B^*$, then

$B^* \setminus b_k$ is optimal sol$^n$

for $\{b_1, b_2, \ldots b_{k-2}\}$

Contradiction: let $C \subset \{b_1, b_2, \ldots b_{k-1}\}$ be the optimal set

i.e. $W(C) > W(B^* \setminus \{b_k\})$

# #weighted sum

$n$ items $\{1, 2, \ldots n\}$ with weight function $w : [n] \to R_{>0}$ and $w > 0$

find subset $S \subseteq \{1, 2, \ldots n\}$ such that

$$\sum_{i \in S} w(i) < W$$

$$\sum_{i \in S} w(i) \text{ is max}$$

Subproblem:

wt(i) = optimum weight from 1st i elements

Recurrence:

if $n$ is not in optimum set:

$wt(n) = wt(n-1)$

else:

$wt(n) = w_n + (\quad)$

Subproblem (Revised):

$Wt(i, T)$ : optimum weight from 1st $i$ elements with weight limited to $T$

Recurrence:

at $n$

if $n \notin$ optimum set:

$Wt(n-1, T) = Wt(n, T)$

else:

$Wt(n) = Wt(i-1, W - w_n) + w_n$

# Problem (Knapsack)

→ $n$ items
→ weight function $w : [n] \to R_{>0}$
→ cost function $v : [n] \to R_{>0}$
→ budget $W > 0$
→ find a subset s.t. cost of items is maximised

→ SUBPROBLEM:

$Wt(i, C, T)$ — index: $i$; constraint: $T \leq W$

→ Recurrence:

At $i$,

if $i \notin$ optimal set

$Wt(i, T) = Wt(i-1, T)$

else:

$Wt(i, T) = Wt(i-1, T - w_i) + c_i$

final: $Wt(i, T) = \max\{ Wt(i-1, T-w_i) + c_i,\ Wt(i-1, T) \}$

Algorithm and Running Time

KT: 6.2 & 6.4
Jeff: 3.1 & 3.8