D&C.

Dynamic Programming (DP)

Greedy Algo

Optimal substructure

Optimal solⁿ for a larger instance

in terms of → optimal solⁿ for sub-problems

max/min

choose one possibility

see if you can identify one choice (with some easily computable property) that is guaranteed to be in atleast one optimal solⁿ.

Proof may based

Coin Change Suppose you want to make a change of n Rupees using coins of denomination :- $\underbrace{1, 2, 5}_{\text{sufficient } \#}$

Criterion :⇒ You want to use the min # of coins.

Change[i] : min # of coins need to make a change for i Rs

$i \geq 5$

5 Rs $i-5$

2 Rs $i-2$

1 Rs $i-1$

1, 2, 5, 2, 1, 2, 5

Greedy

DP

$$\boxed{\text{Change}[i] = 1 + \min(\text{Change}[i-5], \text{Change}[i-2], \text{Change}[i-1])}$$

Greedy:
vs
DP

if $n \geq 5$
→ select 5
else if $n \geq 2$
→ select 2
else
select 1

RecChange (n)
if (n>5)
return 1 + RecChange (n-5)

Now, I claim that my greedy algo works
what is the proof?

$\underline{n}$ The greedy algo chooses exactly $t = \lfloor \frac{n}{5} \rfloor$ 5 Rs coins,
Suppose an optimal sol^n chooses $\underline{t' \leq t}$ 5 Rs coins, $\underline{t' > t}$?

Assume $\underline{t' < t}$ $\Rightarrow$ Atleast $5(t-t')$ Rs. have been chaged in the optimal sol^n using 1 Rs & 2 Rs coins

$\rightarrow$ Optimal sol^n uses 3 2 Rs Coins

$\times$ $\rightarrow$ Replace this sol^n with $\underline{5\,Rs + 1\,Rs}$ } better sol^n

Uses $\underline{2+2+1}$ (at least $\underline{1}$ 1 Rs coin)

any comb^n of 5 that $\underline{\text{uses } 1}$

$\times$

$1, 1, 1, 1, 1$ } $\rightarrow$ 5 Rs coin

Replace with 5 — better sol^n

$\rightarrow$ $\underline{t' = t}$

What remains → Sum of no more than 4 Rs

Exhaustive search of all the cases

Greedy :→ how many 1 Rs coin? → atmost 1

2

has to be one of the optimal solns

Greedy way of choosing may not always work.   4, 4 Optimal
↳ 2 coins

Suppose n Rs coins 5, 4, 1.   n=8

will my greedy algo still work?   5, 1, 1, 1 → 4 coins

Greedy ~~X~~

Once you identify a greedy property

1. You need to prove that this gives an optimal sol^n.

or 2. You can give a counter example where this greedy property doesn't give you an optimal sol^n..

Activity Selection :-> Suppose you've a classroom NR 121

Suppose there are n requests

$1, \ldots n$

Each request has a start time $s(i)$ & a finish time $f(i)$

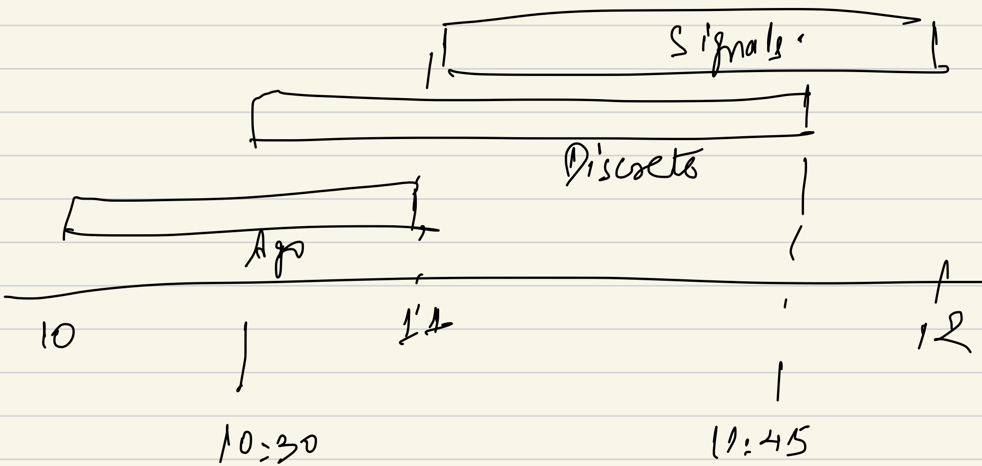
$$\boxed{s(i) < f(i)}$$

10:00 – 11:00 Algo

10:30 – 11:45 Discrete

11:00 – 12:00 Signals & Systems

Compatible



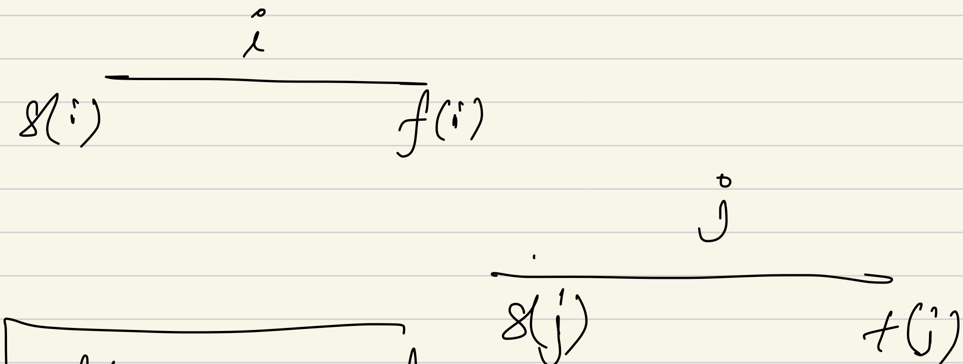
Algo, Signals ✓

Discrete ✓

Algo, Discrete ✗

Discrete, Signals ✗

Two requests i & j are said to be Compatible if they don't overlap



$\boxed{f(i) \leq s(j)}$ or $\boxed{f(j) \leq s(i)}$

Optimization Problem :- Goal is to select a compatible subset of requests that is of max. size.

& do this efficiently.

Naive solution: Take all possible subsets

for each subset

→ check if they're compatible

filter compatible ones

select max size

exhaustive search

exponential-time

Let's define formally

n Request a_1 — — . a_n [sorted in terms of finish time]

$s(1)$ $s(i)$ $f(n)$

$f(1)$ $f(i)$ $f(n)$

T_{ij} :- set of tasks that start after a_i finishes & finish before a_j starts

S_{ij} :→ Optimal sol^n for T_{ij}

subset

(compatible subset of max size)

$$c[i,j] = |S_{ij}|$$

Dummy tasks $\underline{a_0}$ $\underline{a_{n+1}}$

$f_0 \le \min \underline{s_i}$ $\quad$ $s_{n+1} \ge f_n$

Greedy!!

$c[0, n+1]$ is the final sol^n.

DP

$$\underline{c[i,j]} = 0 \text{ if } i=j$$

$$= \max_{i<k<j} \boxed{c[i,k] + 1 + c[k,j]}$$

Greedy:- See if you can identify one choice guaranteed to be in atleast one optimal sol^n.

$\underline{a_0}$ $\underline{a_1}$ $\underline{a_n}$ $\underline{a_{n+1}}$

Proof/ counter example:

1. Select an activity with the shortest time. $(f(i)-s(i))$

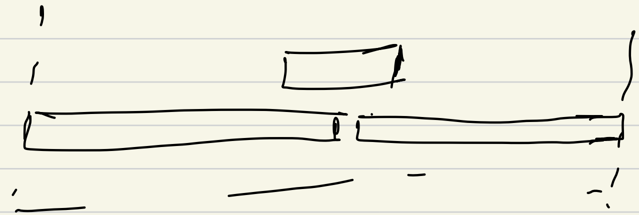
   that'll leave resource available for max amt of time.

2. Select an activity with the earliest finish time. $f(i)$

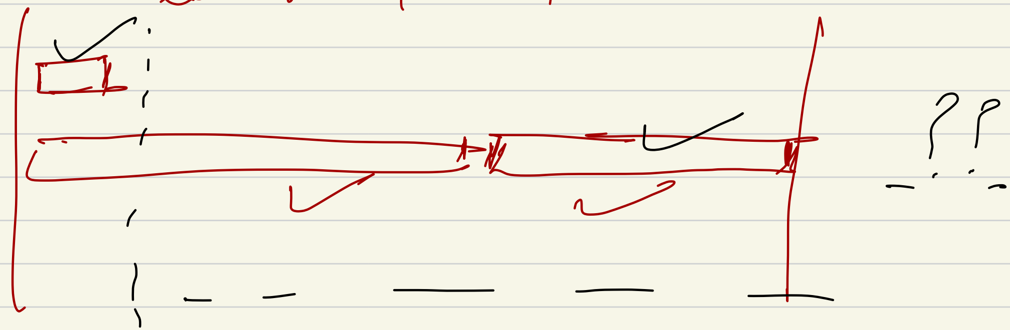
3. Choose an activity with min # of incompatibilities.

   would leave max no. of activities.

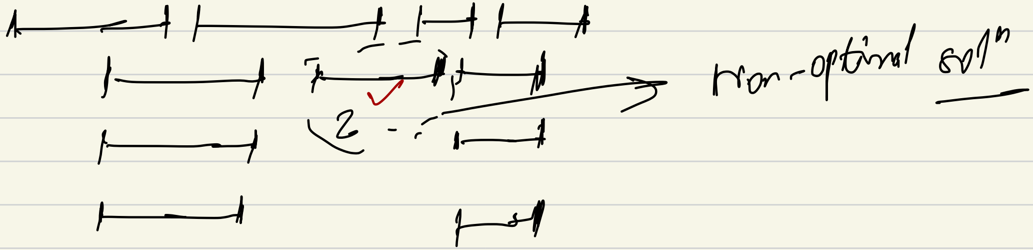
1. activity with the shortest time.



2. Activity with earliest finish time.



3. Activity with min. # of incompatible requests



Choose the activity with the earliest finish time

→ Remove incompatible acts.

→ Continue until all requests are processed.

Proof ⇒ T_{ij} :- set of tasks starting after a_i finishes & finishing before a_j starts

Let a_m be an activity in T_{ij} with the earliest finish time.

$\Rightarrow$ a_m is included in some optimal sol^n of I_{ij}

Proof:$\rightarrow$ Let S_{ij} be the optimal sol^n

$\rightarrow$ a max size subset of the mutually compatible activities in I_{ij}.

[Trick: show if it is different than greedy you can convert it to greedy while remaining optimal]

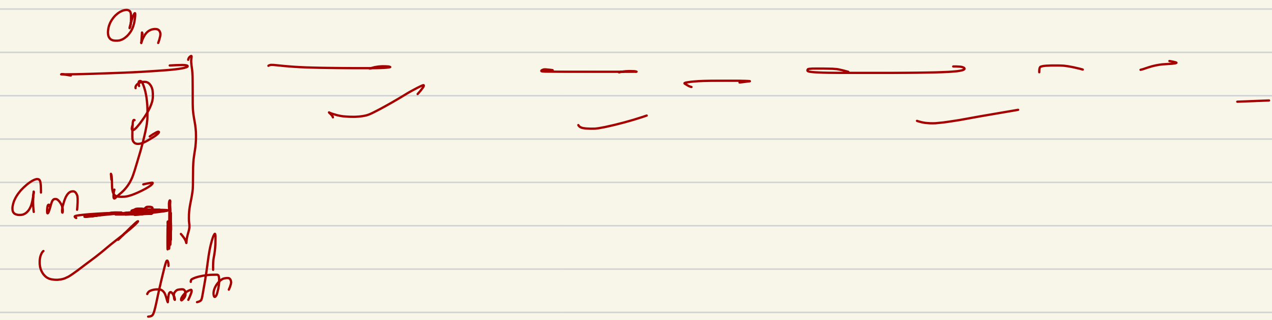
Let a_n be an activity with the earliest finish time in S_{ij}

Pf $a_n = a_m$: we're done

Pf $a_n \neq a_m$: $\Rightarrow$ $f_n \geq f_m$

Let's take $S_{ij}' = S_{ij} - \{a_n\} \cup \{a_m\}$ — still compatible? — size same? ✓

S_{ij}



$\Rightarrow$ S_{ij}' activities are disjoint

Proof.

Pseudo Code: → Let's assume that the activities are sorted as per the finish time, let s & f be the two arrays
start finish.

Greedy-activity-selector (s, f)

```
n = s.length
A = {a_1}
k = 1        // last activity selected
```

linear time $O(n)$

* sorted as per finish times

$O(n \log n)$

```
for m = 2 to n
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

Problem → Single resource & set of n requests to use the resource for an interval of time

i $S(i)$ $f(i)$

10:00 11:00

run my code on a server

t_i – time (length of job)

contiguous time interval

d_i deadline

Resource is available from time S.

Different requests must be assigned non-overlapping intervals.

Goal → Suppose we want to satisfy each request.

✓ 2hrs | 2

1.2hr

4 5

but we're allowed to

let certain requests run late.

s

$s(i)$ — t_i — $f(i) = s(i) + t_i$

schedule, all the request

to be determined by the algo

We say that a request i is late if it misses the deadline, $f(i) > d_i$.

The lateness $l_i = 0$ if $f(i) \le d_i$

$= f(i) - d_i$ o.w.

$\max \{ l_1 \quad l_2 \quad — \quad l_n \} \rightarrow$ minimize

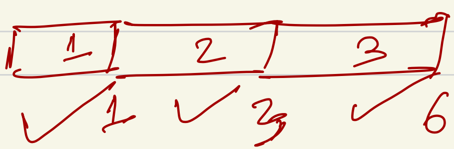
The goal of our optimization would be to schedule all requests, using non-overlapping intervals so as to minimize the maxi. lateness

$$L = \max_i l_i$$

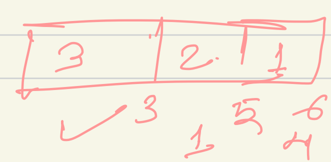
Job1 $t_i = 1$ $d_i = 2$

Job2 $t_i = 2$ $d_i = 4$

Job3 $t_i = 3$ $d_i = 6$



$L = 0$



$L = 4$

We need to look at (t_i, d_i) data

Approach 1 $\Rightarrow$ Schedule the jobs in increasing order of their length: Get the shortest jobs out of the way quickly

X Counter example?

shorter job but deadline is far

vs

longer job but deadline strict

$t_1 = 1$	$d_1 = 100$
$t_2 = 10$	$d_2 = 10$

Greedy: Job 1 ✓

Job 2 $l_2 = 1$

$L = 1$

Optimal: Job 2 ✓

Job 1 ✓

$L = 0$

Approach 2:→ We're concerned about jobs whose available slack time $d_i - t_i$ is very small.

→ Schedule in increasing order of slack time

Counter Example?

job with 0 slack deadline later

job with higher slack deadline earlier

$t_1 = 10$ $d_1 = 10$ Slack = 0

$t_2 = 1$ $d_2 = 2$ slack 1

Greedy

Job 1 ✓

Job 2 9

$L = 9$

Optimal

Job 2 ✓

Job 1 1

$L = 1$

t_i, d_i

Smallest t_i

Smallest $d_i - t_i$

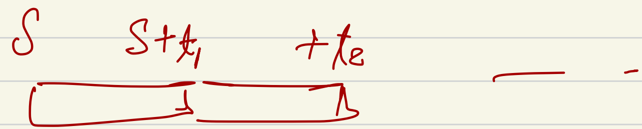
Smallest d_i

Approach 3:- Choose job with the earliest deadline

Sort the jobs in increasing order of deadlines

$d_1 \le d_2 \;--\; \le d_n$

Schedule in this order



no idle time

Finish time $\quad S + \sum_{i=1}^{n} t_i$

Optimal sol^n schedule

Schedule A' has an inversion if a job i with deadline d_i is scheduled before another job j with deadline $d_j < d_i$

$$\left.\frac{i}{d_i} > \frac{j}{d_j}\right\} \text{inversion}$$

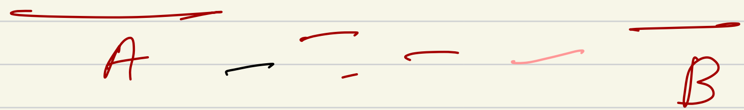
By def^n, greedy algo doesn't have any inversion.

* There is an optimal schedule with no inversion.

↓ change it to greedy without comp. optimality.

Let O be an optimal schedule.

If O has an inversion, there is a job A scheduled before job B



$$d_A > d_B$$

If we advance in the scheduled order of jobs from A to B, there has to be a point at which the deadline decreases for the first time

$\Rightarrow$ Consecutive pair of jobs i j

(i, j) pair of jobs (inverted)
consecutive

$d_i > d_j$

i scheduled before

If we swap (i, j) $\Rightarrow$ eliminate this inversion.

$\rightarrow$ The new schedule has max lateness no greater than that of O. (optimal schedule)

Proof: Schedule O Assume each request r is scheduled $[s(r), f(r)]$ lateness l_r

$$L = \max_r l_r$$

Let O' denote the schedule where I swap (i, j)

$[\bar{s}(r), \bar{f}(r)]$ $\bar{l}_r$ $\bar{L} = \max_r \bar{l}_r$

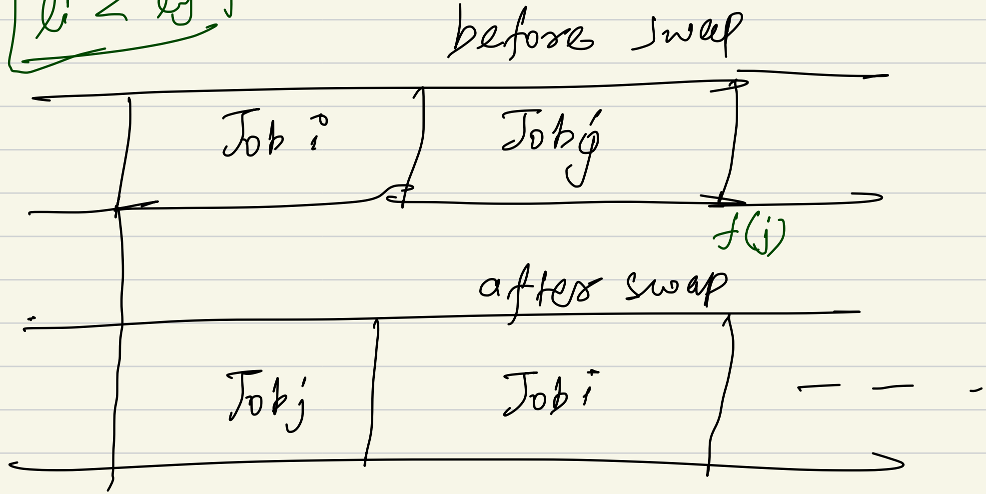
$$\boxed{\begin{aligned} l_j &= f(j) - d_j \\ \bar{l}_i &= f(j) - d_i \end{aligned}}$$

i, j

$d_j \quad d_i$

$d_i > d_j \Rightarrow \boxed{\bar{l}_i < l_j}$

* Job j finishes earlier in the new schedule $\Rightarrow \boxed{\bar{l}_j < l_j}$



$L = \{ \ldots l_i, l_j, \ldots \}$

$\bar{L} = \max\{ l_1 \ldots \bar{l}_i, \bar{l}_j, \ldots l_n \}$

$\bar{l}_j < l_j$

$\bar{l}_i < l_j$

* All jobs other that i & j finish at the same time in both schedules

$\bar{l}_r = l_r$ if $r \neq i, j$

$L = \max\{ l_1, \ldots l_i, l_j, \ldots l_n \}$

$\bar{L} = \max\{ \bar{l}_1 \ldots \bar{l}_i, \bar{l}_j, \ldots \bar{l}_n \}$

$\Rightarrow$ swap doesn't increase L

inversion $\Rightarrow$ swap $\Rightarrow$ the sol^n is still optimal

O ——— G

Scheduling with deadlines

Simple greedy algo

sort by deadlines

schedule in increasing order

$$\underline{i} < \underline{j}$$

$$d_i > d_j$$

inversion

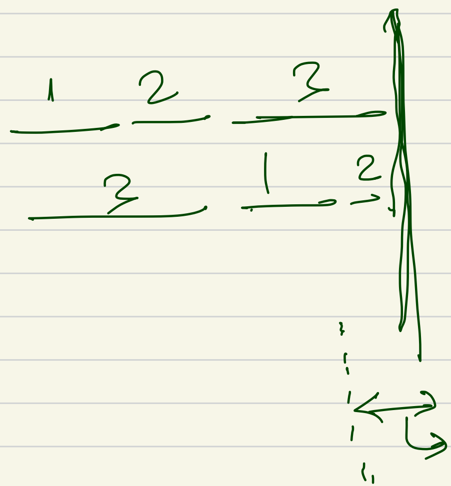
$\longrightarrow$ $\exists$ Optimal schedule with no inversions

multiple reqs with same deadline

3 req. with same deadline

Put in any order

will not change max. lateness



max. lateness will remain the same

Knapsack Problem :-

Setting :- A thief trying to rob a shop, has a knapsack

Various items $1, \ldots n$

ith item — weight w_i value v_i

these are integers

Knapsack :- can carry atmost W weight

Optimization Problem :- Carry as valuable a load as possible that he can fill in knapsack.

What items should he carry?

Ex. 3 items

1	2	3
10 kg	12 kg	3 kg
50$/kg	45$/kg	30$/kg
500$	540$	90$

Knapsack Capacity $\boxed{15\ kg}$

Either take or leave an item

0-1 Knapsack :->

12	3	630$
10	3	540$

Fractional knapsack :-> The thief can take fraction of items

2 : 12 kg
3 : 3 kg → 630$

Is there a better alternative?

1 : 10 kg 2 : 5 kg 725$

fractional knapsack → Greedy Strategy

- Compute the value per kg v_i/w_i for each item
- Sort the items as per this value/pound ration
- The thief begins by taking as much as possible of the item with the greatest value/pound.
- If the supply of this item is exhausted & he can still carry more, he takes as much as possible of the item with the next greatest value/pound & so forth, until he reaches his weight limit W.

Proof :- Given a set of n items $\{1, \dots n\}$

Assume that they're sorted as per value/weight ratio ρ

$$\rho_1 \geq \rho_2 \dots \geq \rho_n$$

Let the greedy sol^n be $G = \langle x_1, \dots x_n \rangle$

where x_i indicates the fraction of item i.

$\in [0, 1]$ $\qquad i - v_i, w_i$

Knapsack capacity $= W$

$$\sum_{i=1}^{n} x_i w_i = W$$

Assume an optimal sol^n $O = \langle y_1, \dots y_n \rangle$

$$\sum_{i=1}^{n} y_i w_i = W$$

Prove that G is also an optimal sol^n.

$$G = \langle x_1 \;-\; - \quad x_n \rangle$$

$$O = \langle y_1 \;-\; - \quad y_n \rangle$$

Consider the first item $\underline{i}$ where the two sol^n's differ.

$\boxed{x_i > y_i}$?

By def^n G takes as large fraction of i as possible.

——— x_i — ———

O ——— y_i

O' ——— x_i

Let $\Delta = (x_i - y_i)$

Consider a new sol^n O' from O s.t.

for $j < i$ $\quad y_j' = y_j$

$$y_i' = x_i$$

$\Rightarrow$ I've to now remove items of weight $\underline{\Delta w_i}$ from $\underline{i+1 \to n}$ resetting $\underline{y_j'}$s

Now, argue that the total value in sol^n O' is $\geq$ that of O,

$\Rightarrow$ O' is an optimal sol^n

(Contradiction)

weight added Δw_i removing Δw_i from the later items

i

diff value Δv_i $-$ $\left[\sum_{k=i+1}^{n} t_k v_k\right]$

$= \Delta w_i p_i$

s.t. $\sum t_k w_k = \Delta w_i$

$\sum_{k=i+1}^{n} t_k w_k p_k \leq \sum t_k w_k p_{i+1}$

$\leq \Delta w_i p_{i+1}$

$\leq \sum_{k=i+1}^{n} t_k v_{i+1}$

diff value $\geq \Delta w_i (p_i - p_{i+1})$

greedy ≥ 0

i

{Greedy is also an optimal sol"}

0-1 Knapsack

Knapsack has a capacity W

as full as possible using a subset of items $\{1, \ldots n\}$

Think in terms of DP

$\{1, \dots n\}$

$OPT(i)$: Sol^n using a subset of requests from $\{1, \dots i\}$

$OPT(n)$

$$\begin{cases} n \notin O & OPT(n) = OPT(n-1) \quad \underline{W} \\ n \in O & OPT(n) = v_n + OPT(n-1) \\ & \underline{W - w_n} \end{cases}$$

Missing weight

Also take into consideration the available weight.

$OPT(i, w)$: Sol^n using a subset of items from $\{1, \dots i\}$ subject to an available weight $\underline{w}$

maximum allowed

$$= \max_{S} \sum_{j \in S} v_j \qquad S \subseteq \{1, \dots i\}$$

subject to $\sum_{j \in S} w_j \le W.$

$$\max_S \sum_{j \in S} v_j$$

Recurrence for $OPT(i, w)$

If $w < \underline{w_i}$ $\quad OPT(i, w) = OPT(i-1, w)$

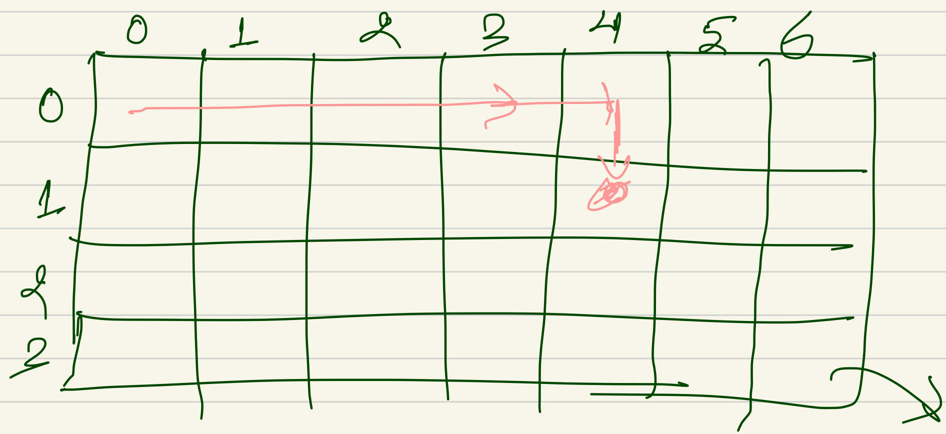
o.w. $\quad = \max \begin{cases} v_i + OPT(i-1, w-w_i) \\ OPT(i-1, w) \end{cases}$

The max. profit $\underbrace{P[n][W]}$

time Complexity $\Theta(\underline{nW}) \leftarrow$ time

3 items	1	2	3
w_i	2 kg	2 kg	3 kg
v_i	4	3	4

$W = 6$ (knapsack capacity)



$OPT(i, w)$

$OPT(i-1, w)$

final answer

fill row wise

Coin change

Interval Scheduling — fix start & finish time

— deadline

fractional knapsack

↳ 0/1 knapsack DP