- **Subset problem** (Similar to Knapsack).

Items : $1, 2, \ldots, n$ — Find a subset of maximal weight,

Weights : $w_1, w_2, \ldots, w_n$ — under W.

(Assume weights are integers).

We want a subset $\{i_1, \ldots, i_k\}$ s.t $\sum_{j=1}^{k} w_{i_j} \leq W$ and $\sum_{j} w_{i_j}$ is maximised and maximal.

→ Either item 1 is a part of Optimal subset Opt.

↳ Opt ($[2,n]$, $W - w_1$)

or not

↳ Opt($[2,n]$, W) .

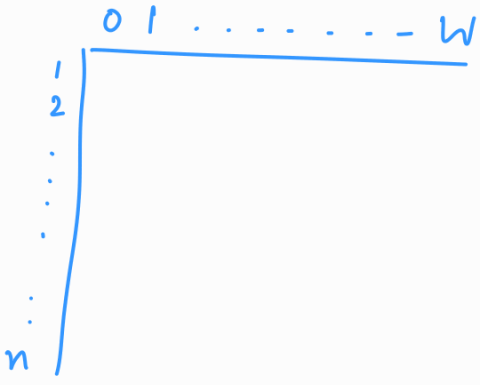
Jeff E.'s book.

↳ 2 parameter DP.

Memoisation :

Structure of problem : Opt($[i,n]$, $\tilde{W}$)

↳ Non-negative

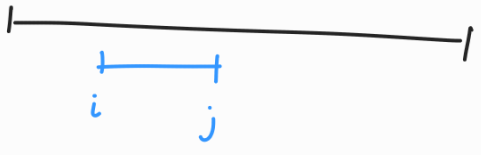


Space complexity : $O(W * n)$.

lookups per entry = 2

- **Subset sum problem.**

Is there a contiguous subset that sums up to A ?

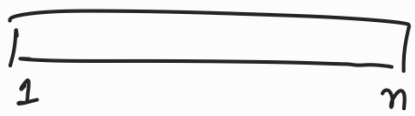


Sliding window.

- **Array partitioning problem:** (Sundar Vishwanathan's notes, IITB).

Also leetcode.

Array A .



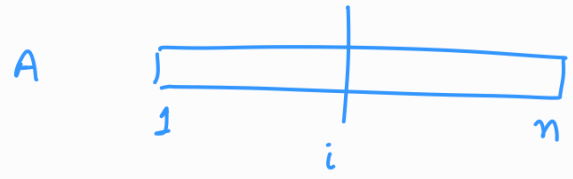
Input is the penalty matrix. $(n \times n)$. $\rightarrow P$

Each partition involves a cost c.

Task: Divide the array into parts s.t each part has contigious elements and total partition cost is minimized.

↳ Sum of penalties of the parts and cost of partitioning

→ If there is only one part, penalty is $P(1,n)$.



↳ Total cost = $P(1,i) + c + p(i+1,n)$. ∵ one partition

No. of partitions

$i_1, i_2, \ldots, i_k \rightarrow c.N + p(1,i_1) + p(i_1+1, i_2) + \ldots + p(i_k+1, n)$.

↓

Have to minimise this.

One of the sol^n:

$$cost([1,n]) = \min\{ \min_k \{ cost([1,k]) + c + cost([k+1,n]) \}, p[1,n] \}.$$

↳ 2D DP. Space comp. $= O(n^2)$

Time comp. $= O(n^2 \cdot n)$ → Actually n^3 largest no.

(OR)

$$MinCost([1,n]) = \min\{ \min_{1 \le k \le n} \{ P[1,k] + c + MinCost([k+1,n]) \}, P[1,n] \}$$

↳ 1D DP

Space: $O(n)$

Time: $O(n^2)$.

↓

Subproblem: $MinCost([i,n])$

$$= \min\{ \{ P[i,k] + c + MinCost([k+1,n]) \mid i \le k < n \} \cup \{ P[1,n] \} \}$$

Proof of correctness for 2nd approach.

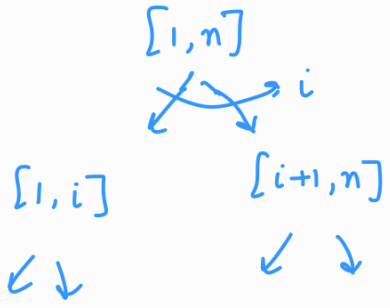
$$cost([1,n]) = \min\{\min_k\{cost([1,k]) + c + cost([k+1,n])\}, p[1,n]\}.$$

Say there is an i^* in opt. partition s.t $[1, i^*]$ is a partition.

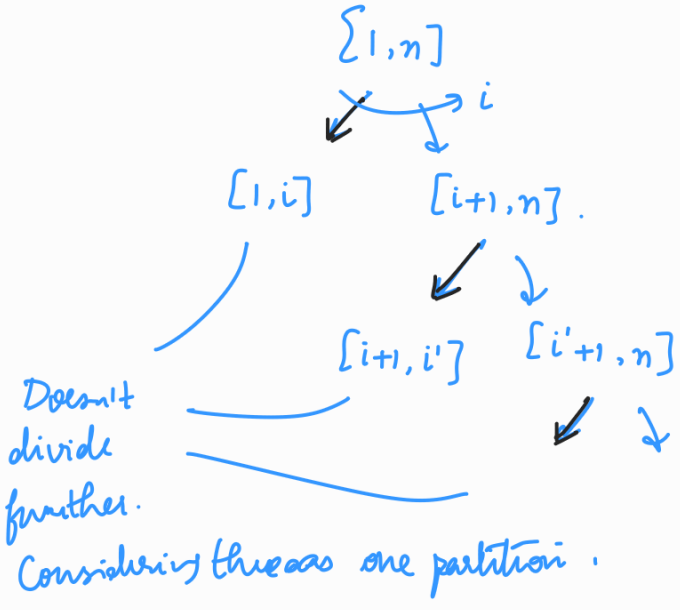
This partition is considered in the second approach as is.

So we're not missing any cases.

1st approach:



2nd approach.



Interval Scheduling:

Jobs/Reqs.	J_1,	J_2, ...,	J_n
Start	s_1	s_2	s_n
Finish	f_1	f_2	f_n

What we did before

Max. no. of jobs that can be scheduled without overlaps.

Want: The duration of scheduling to be maximised as well.

Approach: Take a job or not take it.

→ Sort jobs/req. in increasing order of start time.

DP: Jeff E ☆☆☆

$J_1, J_2, \ldots, J_n$.

Index of First non-overlapping job

If we take J_1, then $|J_1| + Opt(First[1], n)$.

If we don't, then $Opt(2, n)$.

BINARY SEARCH over finish time & start time

$$\therefore Opt(1,n) = \max \begin{cases} Opt(2,n) \\ |f_1 - s_1| + Opt(First[1], n) \end{cases}$$

$$Opt(i,n) = \max \begin{cases} Opt(i+1,n) \\ |f_i - s_i| + Opt(First[i], n) \end{cases}$$

Quiz-2
2nd Question
Go thru soln.