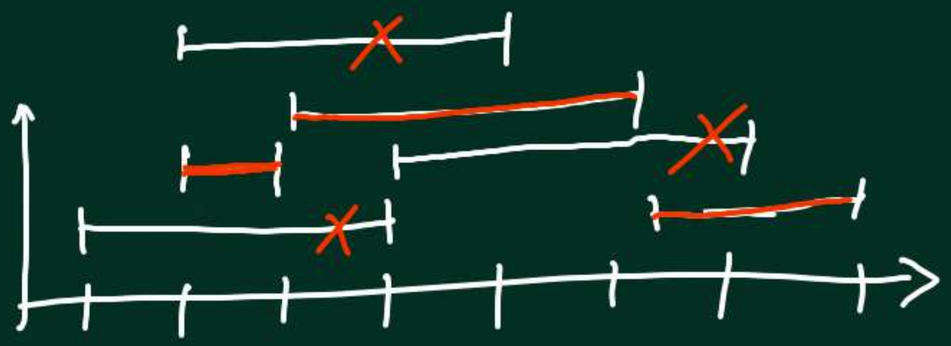
Interval Scheduling

Given a set of intervals, find the largest subset of disjoint intervals.



Greedy approach: min finish time

$\mathcal{I} \leftarrow$ input set

$I_0 \leftarrow$ min finish time $\mathcal{I}$ — sort w.r.t. finish time

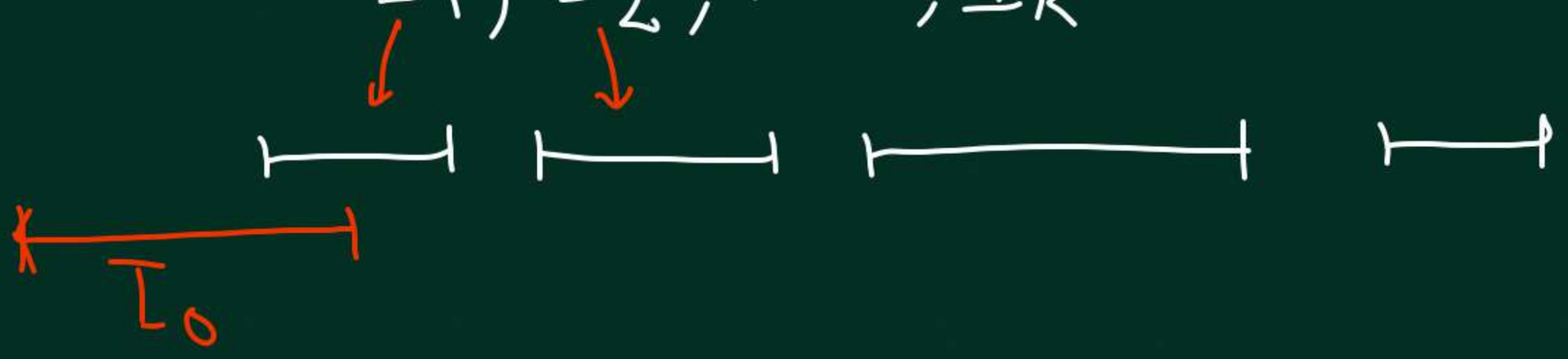
$\mathcal{I}' \leftarrow \mathcal{I} - \{$intervals intersecting with $I_0\}$

Output $I_0 +$ GreedyAlgo$(\mathcal{I}')$

Claim: There always exists an optimal solution that contains I_0 (earliest finish time)

Proof: Consider some optimal solution

$I_1, I_2, \cdots, I_k$



New solution $I_0, I_2, I_3 \cdots I_k$

This is a valid solution. K intervals.

Claim: Greedy algorithm gives an optimal solution.

Proof (induction on number of intervals in the input)

Base case: $n = 1$. Clearly optimal.

Induction Hypothesis: for any set of $n-1$ intervals the greedy gives an optimal so

Induction step:

Greedy · optimal for n intervals.

$\mathcal{I}$ $I_0 \leftarrow$ min finish time $\mathcal{I}' \leftarrow \mathcal{I} - \{$ intersecting I_0

$|\mathcal{I}'| \leq n-1$. Output $I_0 +$ Greedy($\mathcal{I}'$)

Opt($\mathcal{I}$

Claim: $I_0 + Opt(\mathcal{I}')$ is an optimal solution for $\mathcal{I}$

Proof: There is an optimal solution for $\mathcal{I}$ which contains I_0.

Let this be $\underbrace{I_0, J_1, J_2, \cdots J_\ell}_{\ell+1}$

Obs: $J_1, J_2, \cdots J_\ell$ disjoint from I_0

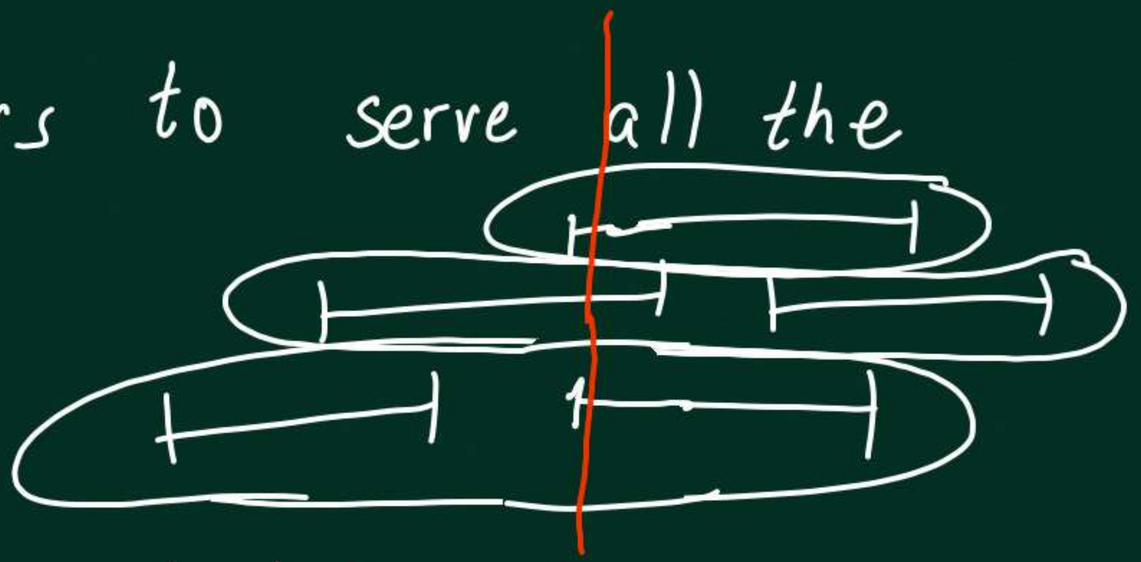
$\in \mathcal{I}'$

$J_1, J_2, \cdots J_\ell \leftarrow$ valid solution for $\mathcal{I}'$

$\ell \leq |opt(\mathcal{I}')| \Rightarrow \ell+1 \leq |I_0 + Opt(\mathcal{I}')|$

Problems

Minimum number of servers to serve all the requests (intervals)

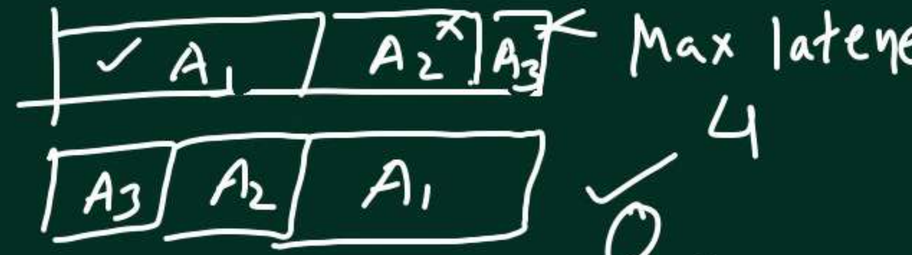


Minimum number of platforms

= Maximum no. trains at a given time instant.

Minimize max lateness

$A_1\ A_2\ A_3$



n Assignments deadlines: $d_1, d_2, \ldots, d_n$

time : $t_1, t_2, \ldots, t_n$

Que: Is it possible to do all assignments within deadlines.

Que: lateness

$l_i = \max(f_i - d_i, 0)$

	A_1	A_2	A_3
time	3	2	1
deadln	6	4	2

Que: Maximize the number of assignments within deadline.

ess

Largest duration Interval scheduling

Given a set of intervals, find a set of disjoint intervals with maximum possible total length

Greedy approaches:

① Max length first

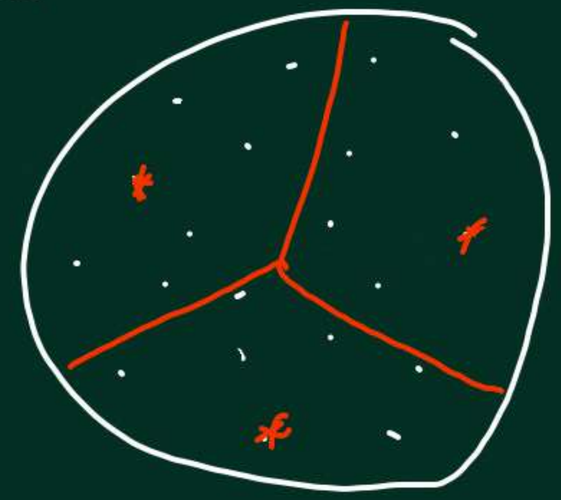
② minimize the intersecting length.

③ minimize the number of intersections.

④ earliest start time ⑤ earliest finish time

Dynamic Programming

- Categorize all possible solutions into various categories
- Find an optimal solution from each category
  using the same algorithm recursively
  on some other input instances.
- Compare these optimal solutions
  and take the best.
- Store the solutions for all the inputs you have already solved.



Longest duration interval scheduling

possible solutions ← subsets of disjoint intervals

I_1 (0, 5) ✗ I_2 (4, 8) ✗ I_3 (3, 7) ✗ I_4 (2, 6) ✗ I_5 (5, 9) ✓

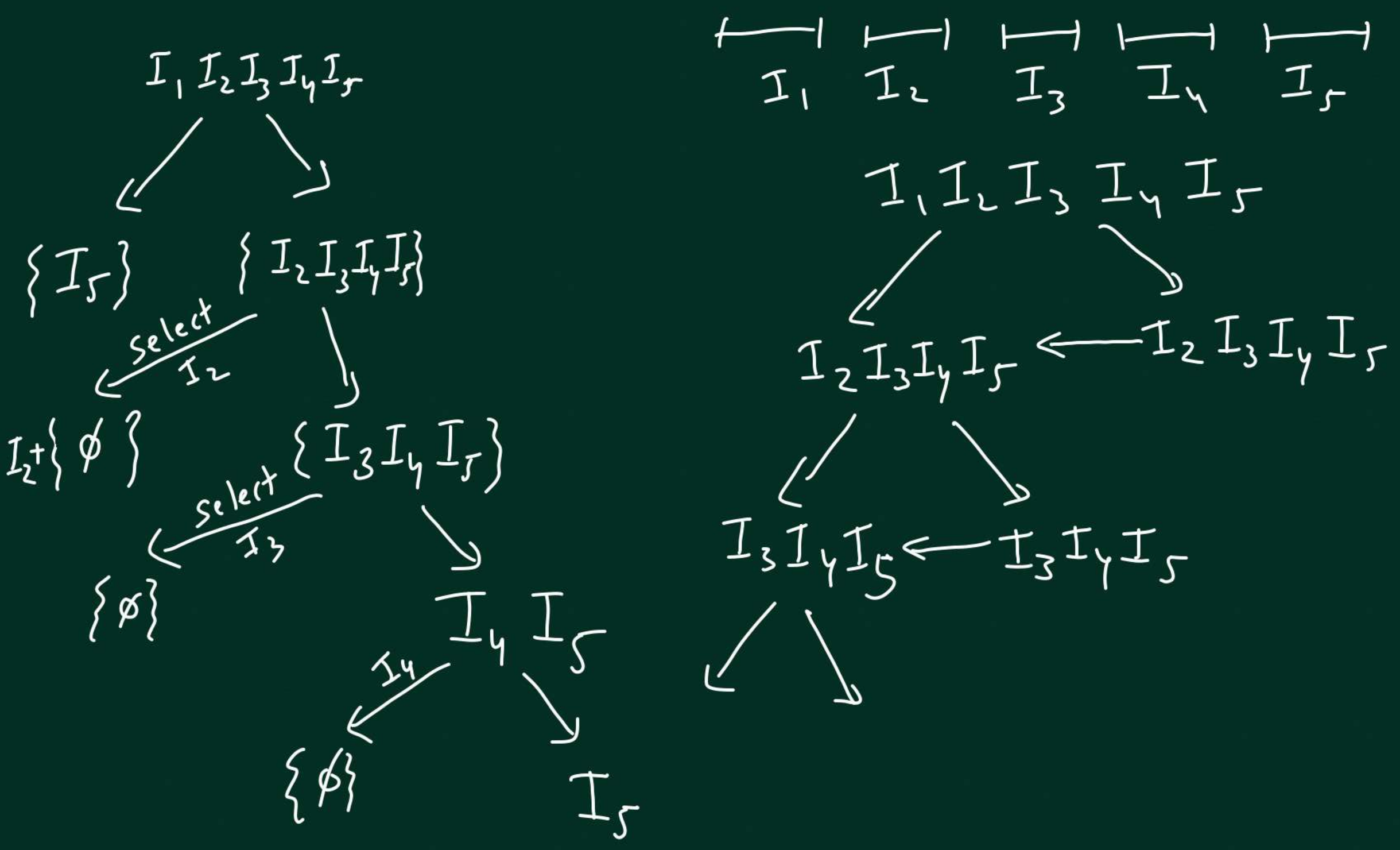
solutions which contain I_1

$ALG(I_5) + I_1$

Solution which don't contain I_1

$ALG(I_2, I_3, I_4, I_5)$

Best


$I_1\ I_2\ I_3\ I_4\ I_5$
$\{I_5\}$
$\{I_2\ I_3\ I_4\ I_5\}$
select I_2
$I_2 + \{\emptyset\}$
$\{I_3\ I_4\ I_5\}$
select I_3
$\{\emptyset\}$
$I_4\ I_5$
I_4
$\{\emptyset\}$
I_5
I_1 I_2 I_3 I_4 I_5
$I_1\ I_2\ I_3\ I_4\ I_5$
$I_2\ I_3\ I_4\ I_5$
$I_2\ I_3\ I_4\ I_5$
$I_3\ I_4\ I_5$
$I_3\ I_4\ I_5$

$I_1 I_2 I_3 I_4 I_5 I_6$

$I_1 \in$ $\quad$ $I_1 \notin$

2^n "distinct recursive calls."

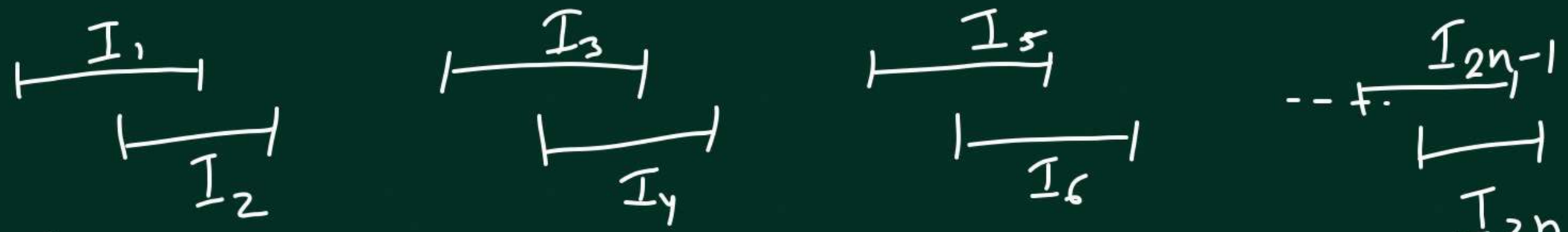
$I_2 I_3 I_5 I_6$ $\quad$ $I_2 I_3 I_4 I_5 I_6$

$I_2 \in$ $\quad$ $I_2 \notin$ $\quad$ $I_2 \in$ $\quad$ $I_2 \notin$

$I_3 I_6$ $\quad$ $I_3 I_5 I_6$ $\quad$ $I_3 I_4 I_6$ $\quad$ $I_3 I_4 I_5 I_6$

2^3 $\quad$ $\emptyset$ $\quad$ I_6 $\quad$ I_5 $\quad$ $I_5 I_6$ $\quad$ I_4 $\quad$ $I_4 I_6$ $\quad$ $I_4 I_5$ $\quad$ $I_4 I_5 I_6$

I_6 $\quad$ I_6



No. of "distinct" recursive calls $O(n)$

$I_1 I_2 I_3 I_4 I_5 I_6$

$I_3 I_4 I_5 I_6$

$I_2 I_3 I_4 I_5 I_6$

$I_5 I_6$

$I_4 I_5 I_6$

$I_3 I_4 I_5 I_6$

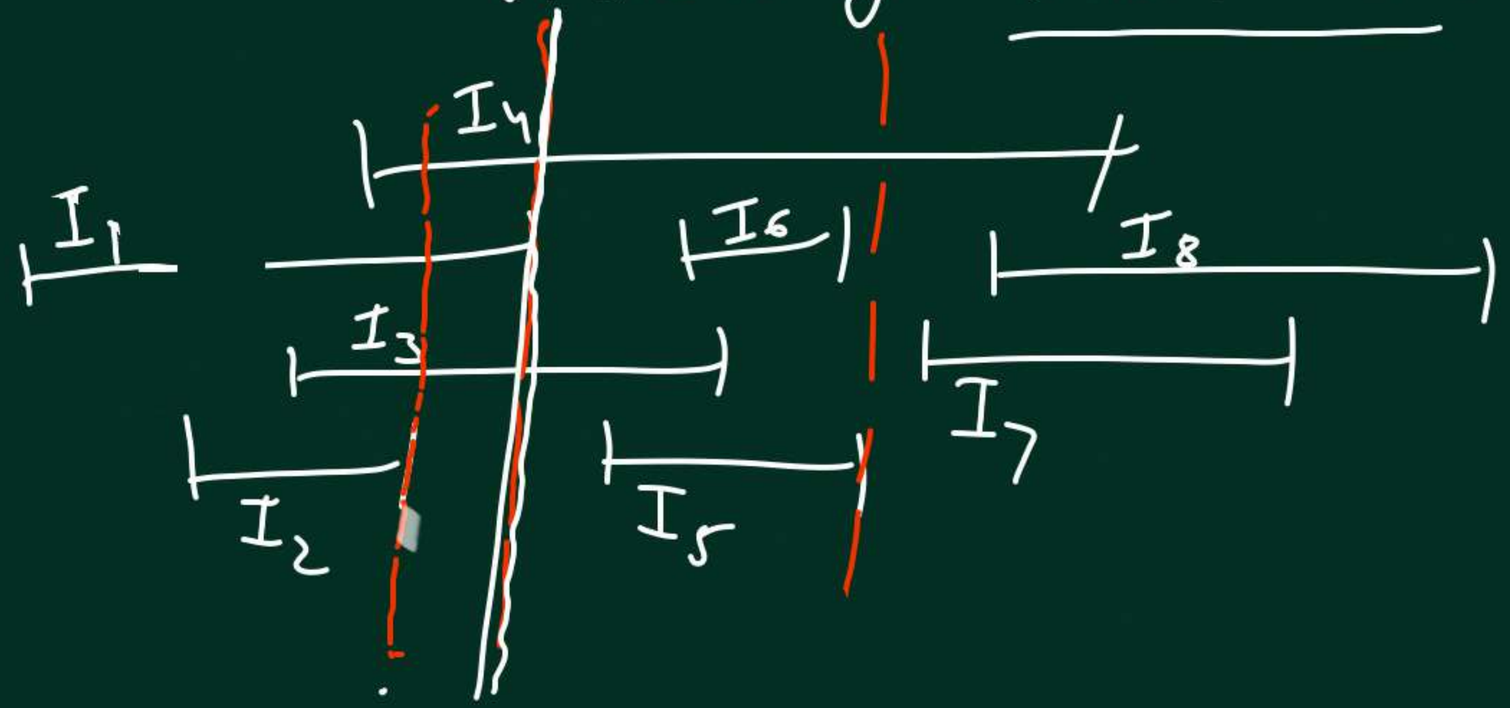
$I_3 I_4 I_5 I_6$

$\emptyset$

I_6

Order — increasing start time

increasing finish time (Bad examples)



At most n distinct recursive calls.

$I_1 \cdots I_n$

$I_j\ I_{j+1} \cdots I_n$ $\qquad$ $I_2 \cdots I_n$

$j \leftarrow$ first index s.t. $start(I_j) > finish(I_1)$

$first[k] \leftarrow$ first index j s.t. $start(I_j) \geq finish(I_k)$

$Opt[j]$ = optimal solution $(I_j, I_{j+1} \cdots, I_n)$

$Opt[1]$

For any k, $\{I_k \cdots I_n\}$

$Opt[k] = \max \begin{cases} Opt[k+1] \\ |I_k| + Opt[first[k]] \end{cases}$

Interval Scheduling.

① Max no. of intervals (Greedy)

② Longest total duration (DP)

$O(n \log n)$

③ Max total weight of selected Intervals (DP)

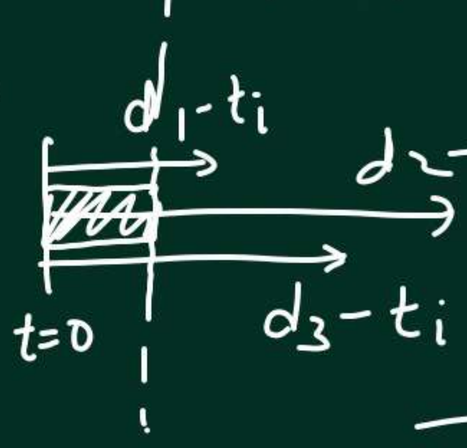
④ Count the number of disjoint subsets of intervals (DP)

Maximum Lateness

n assignments

deadlines $d_1, \cdots, d_n$

time $t_1, t_2, \cdots, t_n$



Minimize max lateness

$L_{max} = \max_i L_i$

Lateness of Assign i

$L_i = \max\{0, f_i - d_i\}$

↑

finish A_i

DP

possible solutions $n!$

n

Suppose A_i is schedulled at the first position.

$$Opt(\{A_1, A_2 \ldots, A_n\})$$

$$\min_i \left\{ \max \left\{ \max\{t_i - d_i, 0\}, Opt(\{A_1, A_2 \ldots A_{i-1}, A_{i+1} \ldots A_n\}) \right\} \right\}$$

$d_1 - t_i$ $d_2 - t_i$ $d_n - t_i$

D D D

No. of "distinct" recursive call

Greedy

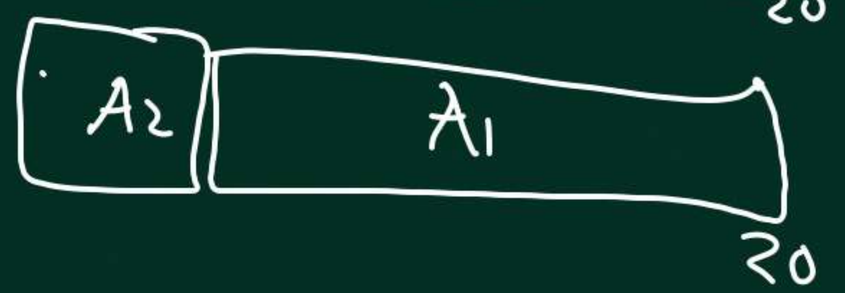
Schedule that assignment first which

1. minimizes d_i
2. ~~minimize~~ t_i
3. ~~minimize~~ $d_i - t_i$

	A_1	A_2
t	18	2
d	19	5
$d-t$	1	3

$L = 15$

$L = 1$ ✓



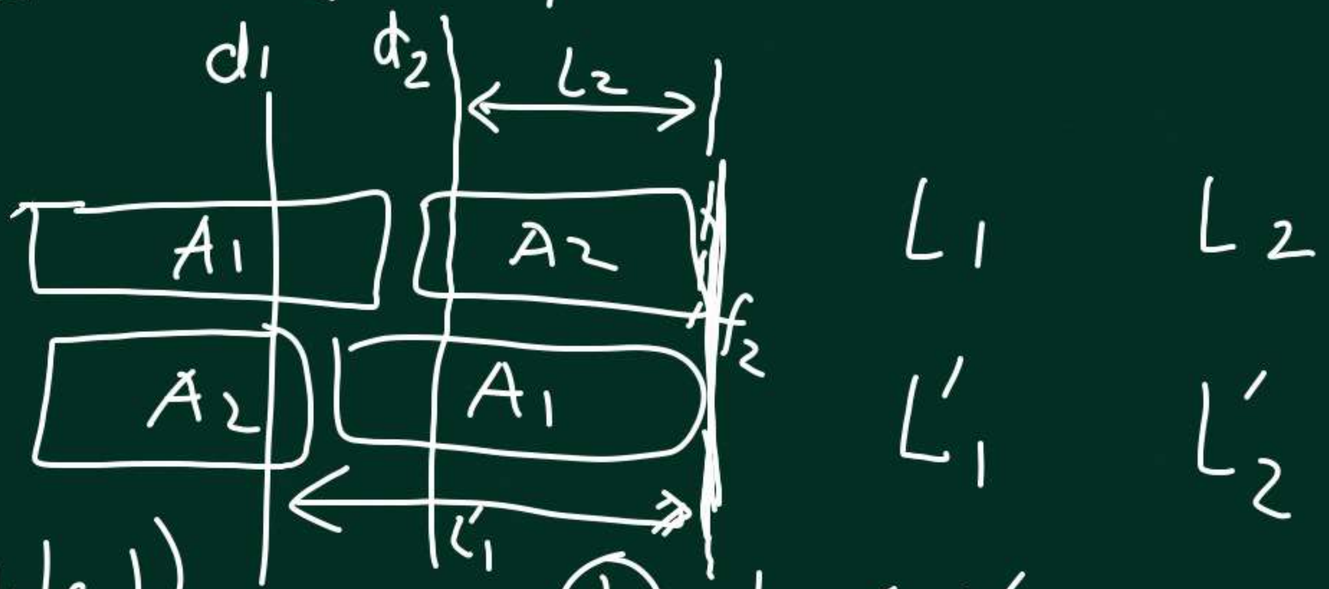
Schedule in the increasing order of deadlines.

Why is this optimal?

$$d_1 \leq d_2 \leq \cdots \leq d_n$$

Two Assignments.

$d_1 \leq d_2$

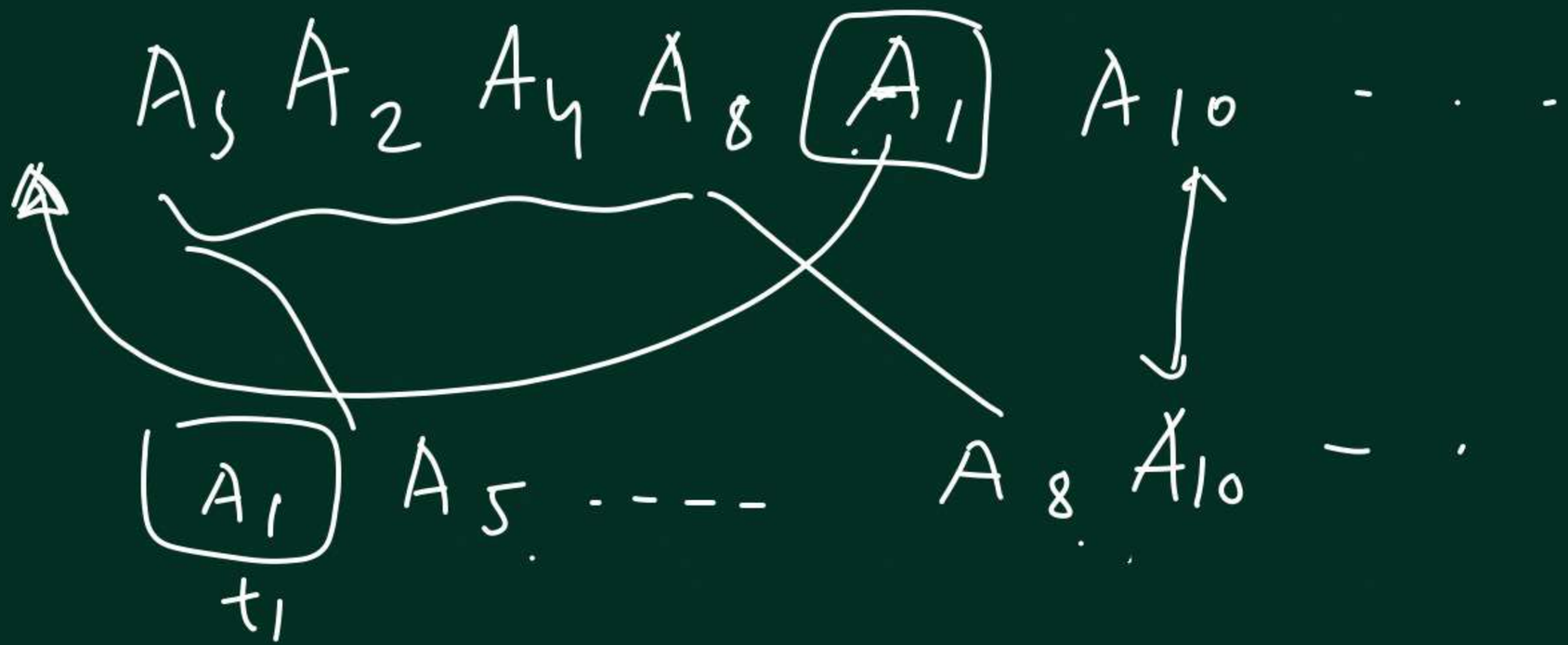


Lateness (Shedule 1)

$\leq$ lateness (schedule 2).

① $L_1 \leq L'_1$

② $L_2 \leq L'_1$



A5 A2 A4 A8 A1 A10 ---

A5 A2 A4 A1 A8 A10

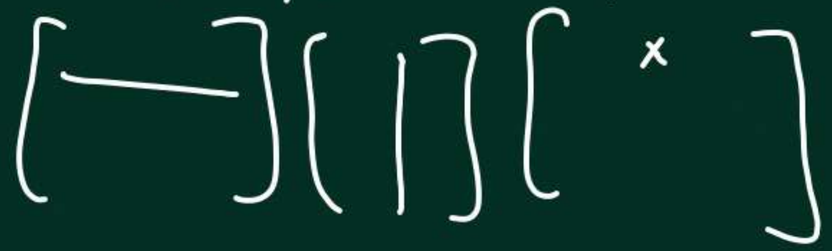
) lateness improves.

Iterated Matrix Multiplication

$$[2\times3][3\times4][4\times5] = 2\times5$$

$$M_1 M_2 M_3 = (M_1 M_2) M_3 = M_1 (M_2 M_3)$$

$$\underset{p\times q}{A}\ \underset{q\times r}{B} = \underset{p\times r}{C}$$



$pr \times q$ multi

$pr\,(q-1)$ additions $\Big\}\ O(pqr)$

$M_1 \quad M_2 \quad M_3$

$2\times3 \quad 3\times4 \quad 4\times5$

2×4

$(M_1 M_2) \cdot M_3$

$24 + 40$

$= 64$

Optimal order?

$10\times3 \quad 3\times4 \quad 4\times5$

$120 + 200 \qquad 60 + 150$

3×5

$3\times4 \quad 4\times5$

2×3

$M_1 (M_2 M_3)$

$30 + 60 = 90$

$P_0 \; P_1 \; P_2 \; P_3$

$P_1 < P_2$

$P_0P_1P_2 + P_0P_2P_3 = P_0P_2(P_1+P_3)$

$P_1P_2P_3 + P_0P_1P_3 = P_1P_3(P_0+P_2)$

$M_1 \quad M_2 \quad M_3 \quad \cdots \quad M_n \equiv M$

$P_0 \times P_1 \quad P_1 \times P_2 \quad P_2 \times P_3 \quad \cdots \quad P_{n-1} \times P_n \quad P_0 \times P_n$

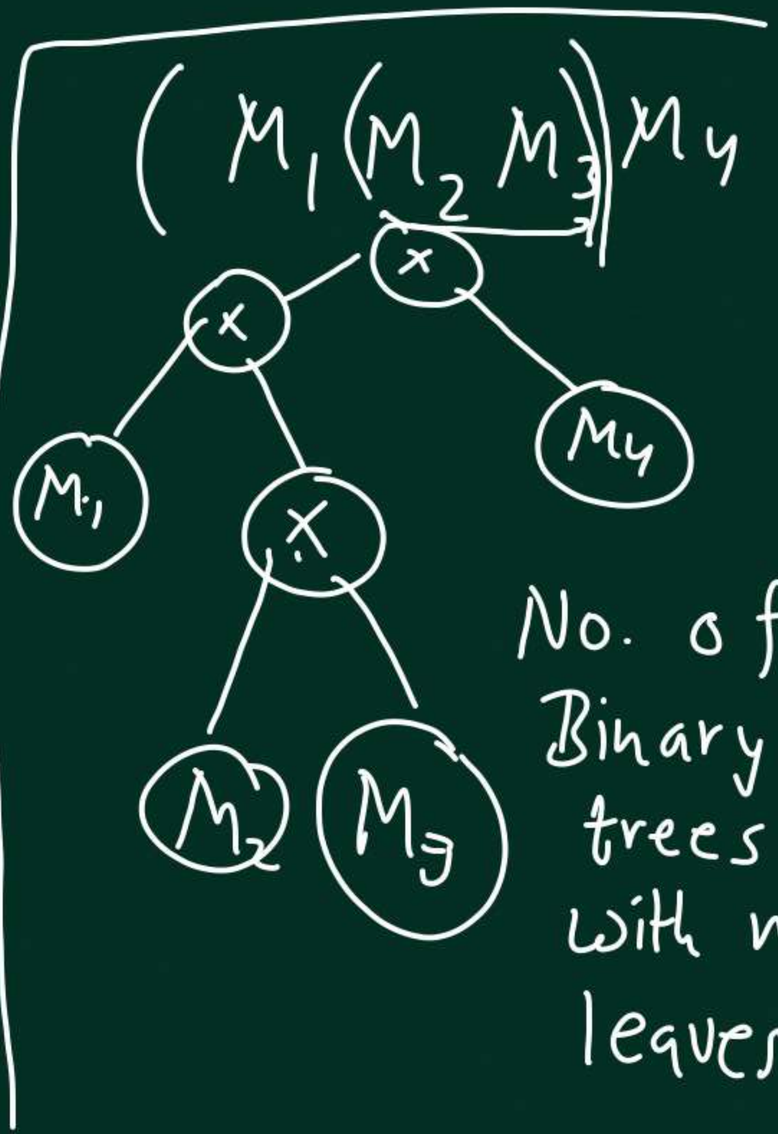
Input $P_0, P_1, P_2 \cdots, P_n$

Output Best order

DP categorizing solutions

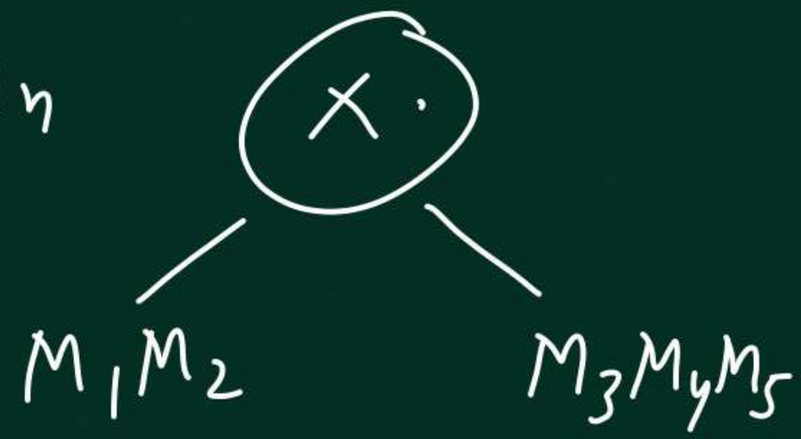
$M_1 (M_2 M_3) M_4$

$M_1 (M_2 M_3 M_4)$



No. of Binary trees with n leaves.

$M_1\ M_2 (M_3\ M_4) M_5 \dots M_n$



Classifying possible orders

① first multiplication

② last multiplication

$M_1\ M_2 \dots M_{i-1} (M_i\ M_{i+1}) \dots M_n \longrightarrow M_1 M_2 \dots (M_i M_{i+1}) \dots M$

$P_0\ P_1\ P_2 \dots P_{i-2}\ P_{i-1}\ P_i\ P_{i+1}$ $\qquad P_0\ P_1\ P_2\ P_{i-1}\ P_{i+1} \dots$

$$Opt(P_0, P_1, \dots P_n) = \min_i \left(P_{i-1} \cdot P_i \cdot P_{i+1} + Opt(P_0, P_1, \dots \right.$$
$$\left. P_{i-1} P_{i+1} P_n \right)$$

$$Opt(P_0, P_1, \cdots P_n)$$

$$= \min_i \left\{ P_{i-1} P_i P_{i+1} + Opt(P_0, P_1, \cdots P_{i-1}, P_{i+1}, \cdots P_n) \right\}$$

No. of distinct recursive calls? $\leftarrow \exp(n)$

$(n-1) \times (n-2) \times (n-3) \times \cdots$

$P_0, P_1, P_2, \cancel{P_3}, \cancel{P_4}, \cdots, P_n$

Every subsequence is possible

$$\underbrace{(M_1 M_2 \cdots M_i)}_{P_0 \; P_1 \quad P_{i-1} \; P_i} \times \underbrace{(M_{i+1} \cdots M_n)}_{P_i \; P_{i+1} \cdots P_n}$$

$Opt(P_0, P_1, P_2 \cdots P_n)$

No of Distinct recursive calls $\leq \binom{n+1}{2}$

$P_3 \equiv$

$P_0 P_1 P_3$

$$\min_i \left[P_0 P_i P_n + Opt(P_0, P_1 \cdots, P_i) + Opt(P_i, P_{i+1}, \cdots P_n) \right]$$

Every recursive call $\longrightarrow$ $P_K P_{K+1} \cdots P_l$

$$Opt(P_k \dots P_l)$$

$$= \min_i \left\{ P_k P_i P_k + \begin{matrix} Opt(P_k \dots P_i) \\ Opt(P_i \dots P_l) \end{matrix} \right\}$$

$O(n^3)$ time.

Implementation

Subset Sum problem.

{6, 2, −5, 4, −7, 15, −20, −10, 8, 9}

Is there a subset whose sum is zero?

Subset Sum problem. (Dynamic Programming)

{6, 2, -5, 13, 4, -7, 15, -20, -10, 8, 9}

-9

Is there a subset whose sum is zero?

Trivial $\leftarrow$ 2^n

Subset $\longrightarrow$ will have a_n

Subset $\searrow$ will not have a_n

Is there a subset of $a_1 \ldots a_{n-1}$ with sum $-a_n$

Is there a subset of $a_1 \ldots a_n$ with sum T

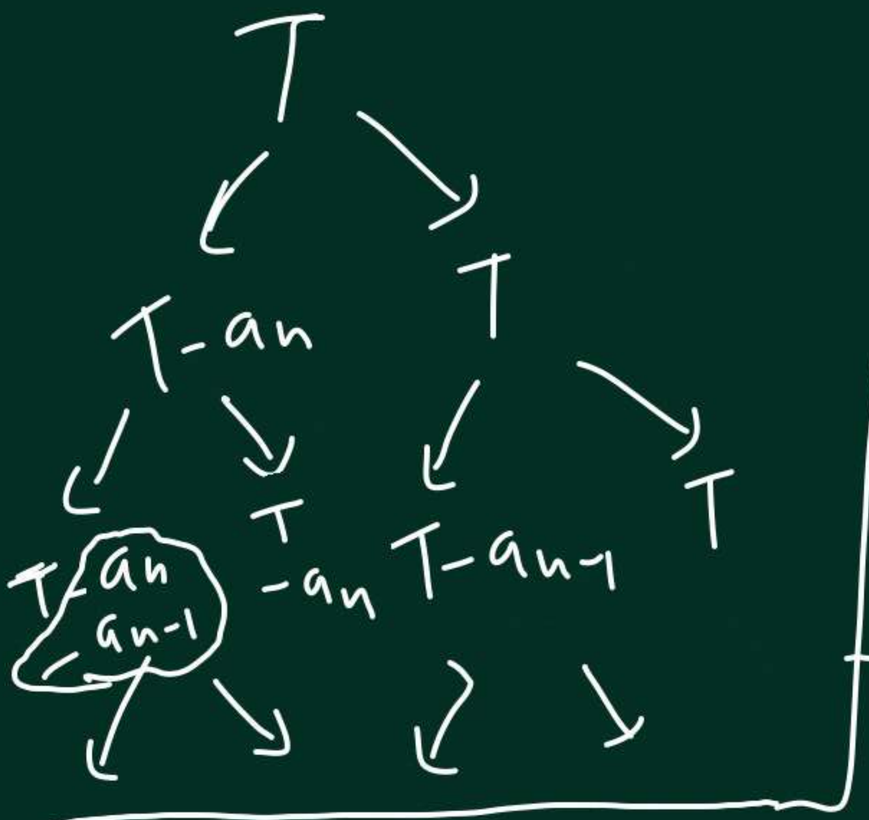
Is there a subset of $a_1 \ldots a_{n-1}$ with sum $T - a_n$

Is there a subset of $a_1 \ldots a_{n-1}$ with sum T.

No. of distinct recursive calls?

prefix $\leftarrow (a_1, \ldots, a_j,$ Target $t)$

n



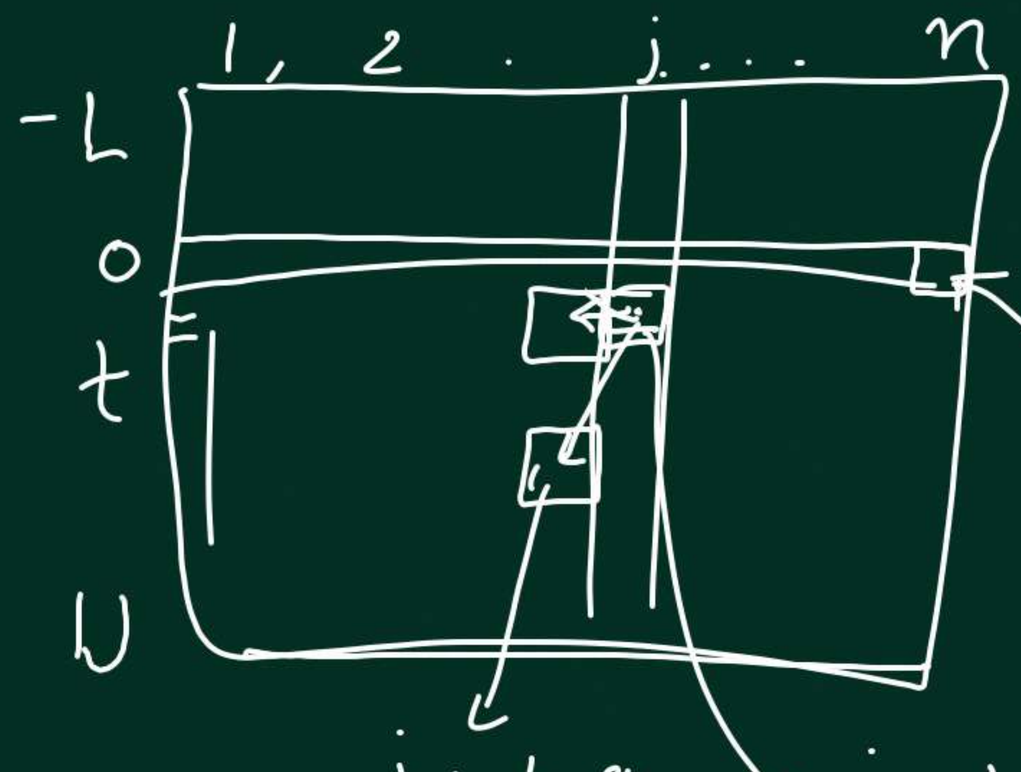
$$\sum_{i:\,a_i<0} \le T \le \sum_{i:\,a_i>0} a_i$$

Size of this range $\sum_i |a_i| + 1$

Subset-sum(j, t) $\Leftarrow$ true (if there is subset of $a_1 \dots a_j$ with sum t

$\Lessdot$ false otherwise

subset-sum$[j, t]$ = subset-sum$[j-1, t]$ OR subset-sum$[j-1, t-a_j]$



Implementation:

n numbers
↓
input size
= total no. of bits

Complexity: $O(n \times \Sigma |a_i|) = O(n \cdot n \cdot 2^l)$

If $a_1 \dots a_n \leftarrow l$ bit numbers pseudo-polynomial time.
input size $n\,l$.

Subset-sum ← NP-hard.

unlikely to have a polynomial time algorithm.

Greedy doesn't work

---

pseudopoly time.

Knapsack Problem:

Value $p_1, p_2 \cdots, p_n$

weights $w_1, w_2 \cdots, w_n$

Pick the subset with max total value but total weight $\leq W$.

$O(n \cdot W)$

$\text{poly}(n, \sum_i p_i)$

Fractional knapsack.

$$x_1, x_2, \ldots, x_n \in [0, 1]$$

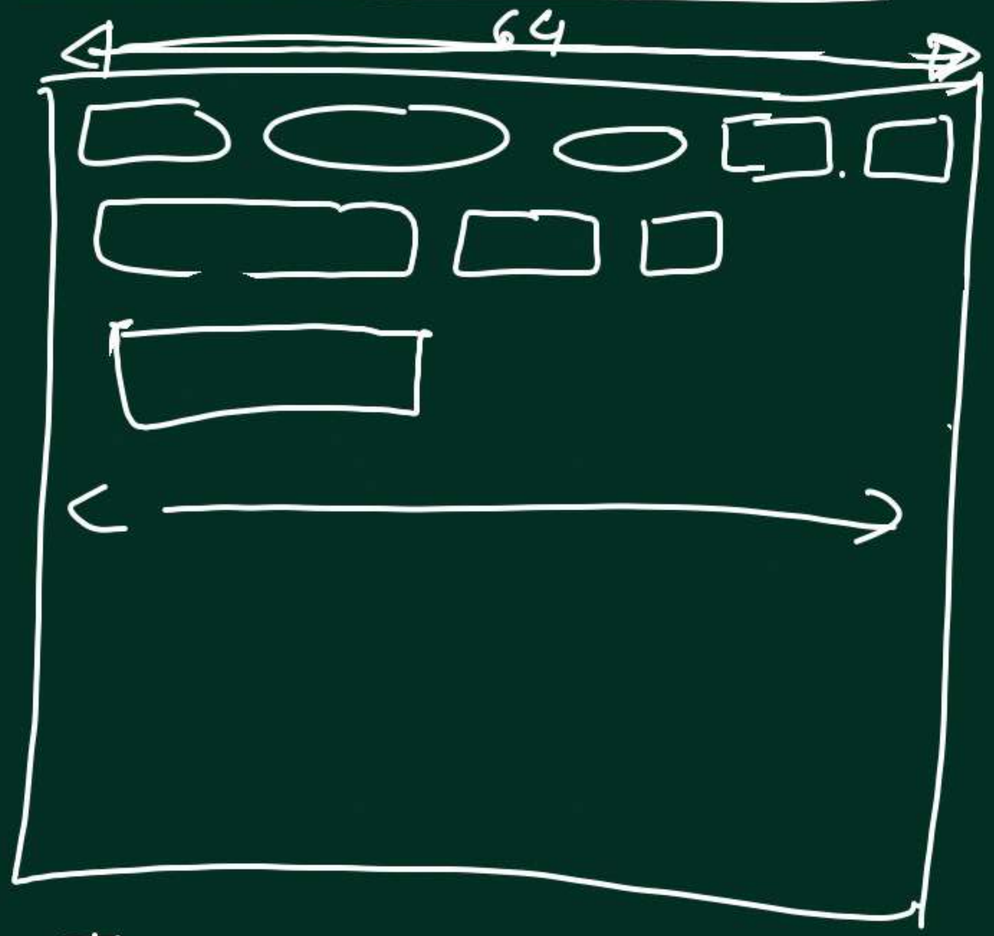
$$\max \sum_i p_i x_i$$

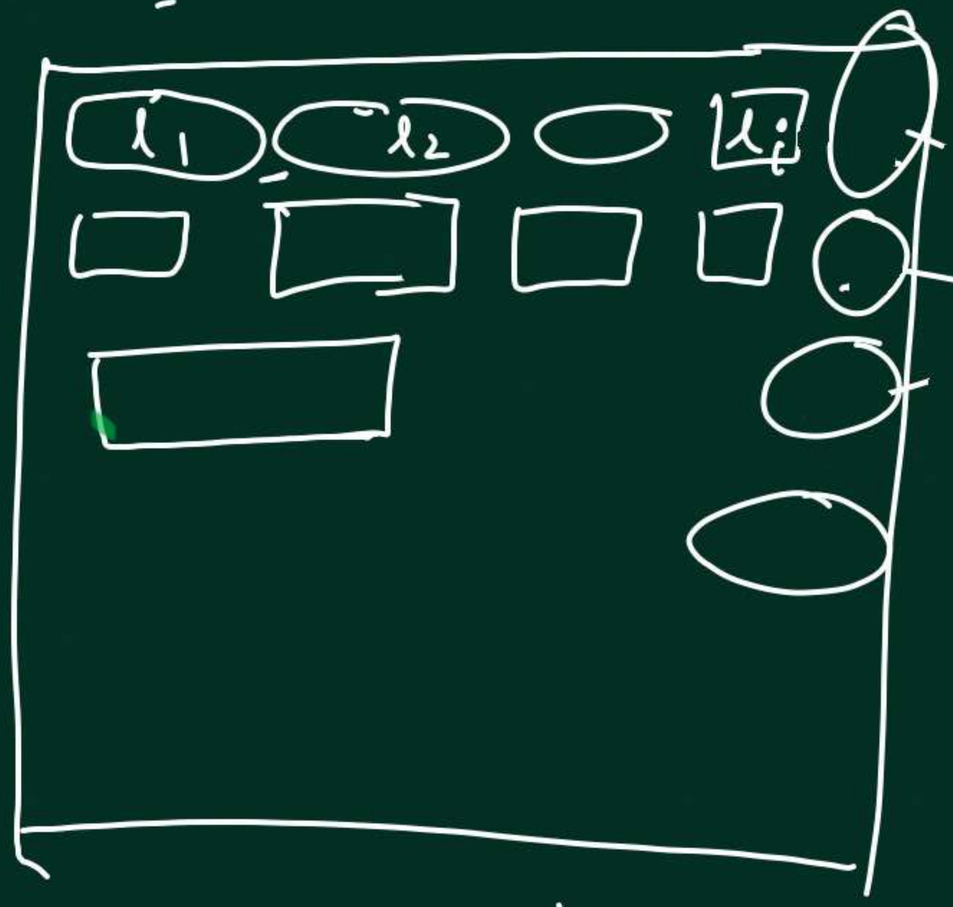
Subject to $\sum_i w_i x_i \leq W$

Greedy algorithm works.

Balanced Margins.

$Slack_i = L - l_1 - 1 - l_2 - 1 \ldots - l_i$





Input ← sequence of word $l_1, l_2 \ldots, l_n$ | limit per line L.

$L \leftarrow$ linewidth

Input : $l_1, l_2, \cdots l_n$

Line k i^{th} to j^{th} word

$$slack_k = L - (l_i + 1 + l_{i+1} + 1 \cdots + l_{j-1} + 1 + l_j)$$

↳ roughly balanced

$\sum_k slack_k = Const.$ | Minimize $\sum_{k=1}^{q} (slack_k)^2$

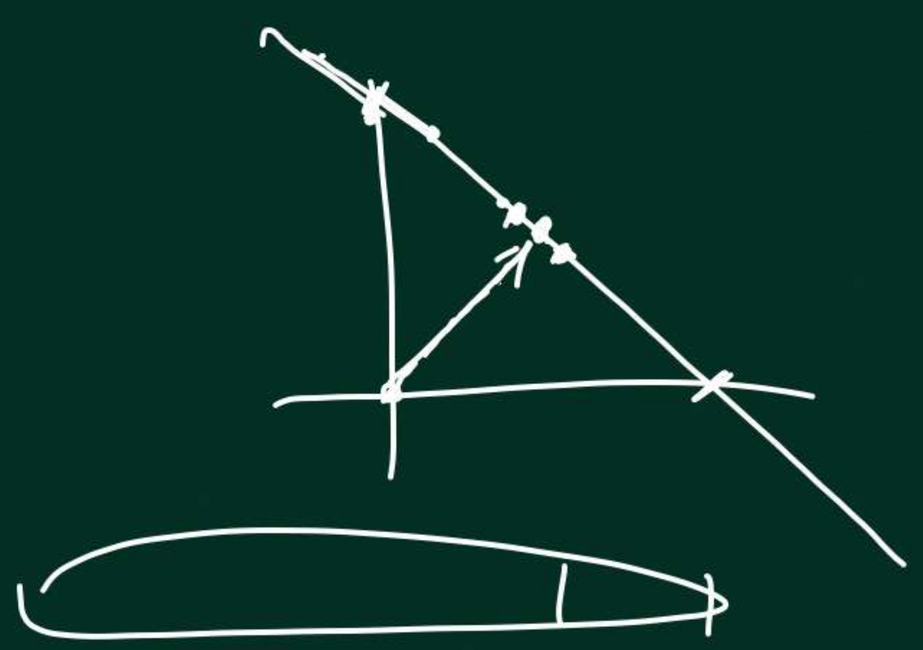
find $a, b, c \in \mathbb{Z}$

$a + b + c = 14$

minimize $a^2 + b^2 + c^2$

$14 = 10 + 2 + 2$ → 108

$14 = 4 + 5 + 5$ → 66



No. of words in line 1

— 1 — line 2 i_1+1, i_2

line 3 i_2+1, i_3;

i_9

} exponentially many possible solutions.

Greedy Ideas.

→ As many as words as possible

→ Average slack

Greedily try to be close to average slack.

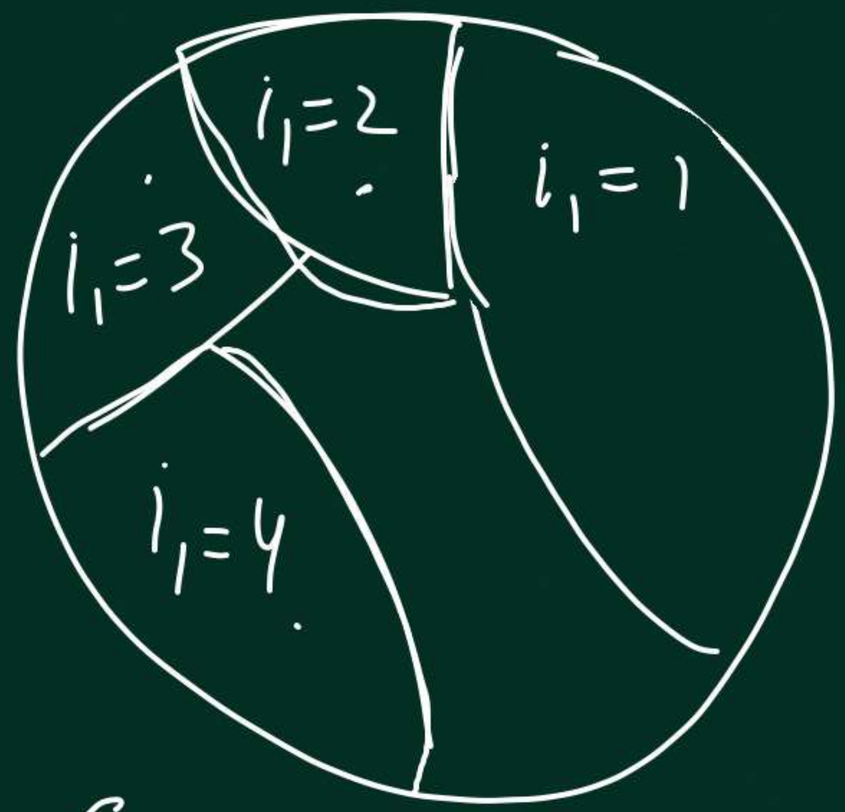
} does this minimize sum of $(\text{slack})^2$?

→ Check two lines at a time, be greedy.

Dynamic Programming

Categorizing Possible solutions

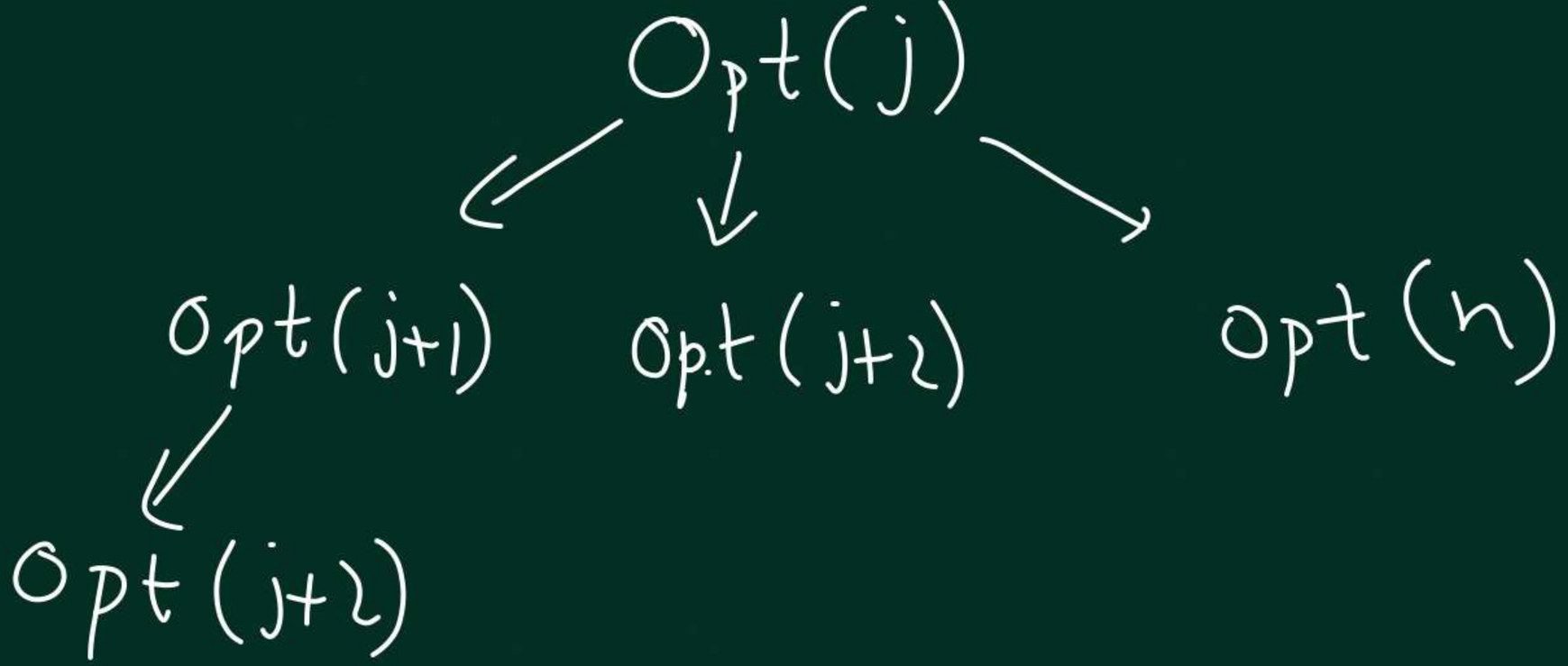
$i_1 \leftarrow$ last word in firs line
$i_2 \leftarrow$ last — second
$i_3 \leftarrow$ last --- third.



$$OPT[1,n] = Min \begin{cases} (L-l_1)^2 + OPT[2,n] \\ (L-l_1-1-l_2)^2 + OPT[3,n] \\ (L-l_1-1-l_2-1 \dots l_h)^2 + OPT[h+1,n] \end{cases}$$

No of "distinct" recursive calls.

Opt(j) ← optimal value for the words $l_j, \ldots, l_n$



for (j=n to 1)

$$Opt(j) = \min \begin{cases} (L-\ell_j)^2 + Opt(j+1) \\ (L-\ell_j-1-\ell_{j+1})^2 + opt(j+2) \\ \vdots \\ \text{---} \qquad\qquad Opt(n) \end{cases}$$

Running time

$O(n^2)$

Implementation

Optimal solution

Compute all slack squares

Before hand $(L-\ell_j-1-\ell_{j+1}-1 \cdots \ell_k)^2$

Edit Distance / Sequence Alignment

Distance between strings.

Spell checker — Most similar, closest

Traning — Training

One way

SNOW
NOWN

4

TREE
RACE

Hamming Distance. = no. of positions where you have diff characters

Levenshtein Distance

= minimum no. of insertions, deletions
or substitutions needed
to go from one string to other

SITTING $\xrightarrow{\text{sub}}$ KITTING $\xrightarrow[I \to E]{\text{sub}}$ KITTENG

↓ Del G

KITTEN

Distance 3

KITTEN

Deletion / Insertion ← cost 2

Substitution

VOWEL - VOWEL ← cost 1

Cons - Cons ← cost 1

VOWEL - Cons ← cost 3

MEAN → NAME

MEAN → NEAN → NAAN → NAMN → NAME

MEAN → MEA → ME → AME → NAME

Del, Del, Inse, Ins

Cost 8

NAME → NEAN → NAAN → NAMN → NAME

1, 3, 3

Cost 8

MEAN → NAME

MEAN → NEAN → NAAN → NAN → NAM → NAME

1, 1, 2, 1, 2

Cost 7

Sequence Alignment.

U G C T G A C U

G A A T G C A

Penalties ← Gap Penalty δ

← Mis Match Penalty

δ α_{CA} δ δ

U G C T G A C U

_ G A T G C A _ _

$\alpha_{CA} + 4\delta$

$x_1\ x_2 \cdots x_n$

$y_1\ y_2 \cdots y_m$

$OPT(n,m) =$ min $\begin{cases} \delta + OPT(n-1, m) \\ \delta + OPT(n, m-1) \\ \alpha_{y_m x_n} + OPT(n-1, m-1) \end{cases}$

DP

① x_n

② y_m

③ x_n

y_m

$OPT(n,m) \leftarrow$ optimal cost of Alignment between $x_1 \cdots x_n$ and $y_1 \cdots y_m$

$OPT(i,j) \leftarrow$ opt cost for alignment between $x_1 \dots x_i$, $y_1 \dots y_j$

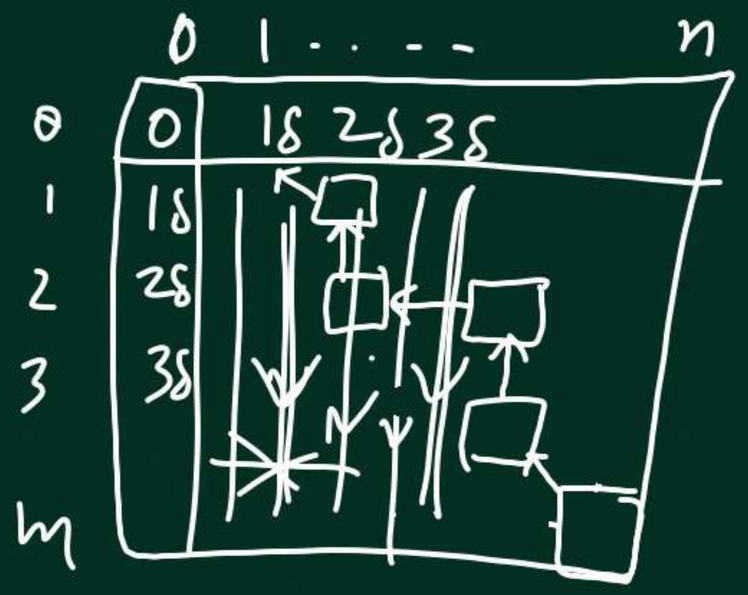
$OPT(i,0) = i \cdot \delta$

$OPT(0,j) = j \cdot \delta$

$OPT(n,m)$

```
for ( i=1 to n)
    for ( j=1 to m)
```

$$OPT(i,j) = \min \begin{cases} \delta + OPT(i-1,j) \\ \delta + OPT(i,j-1) \\ \alpha_{x_i y_j} + OPT(i-1,j-1) \end{cases}$$



Optimal solution?

Opt cost ← space

$O(\min(m,n))$

Space Complexity = $O(mn)$

Space $O(m+n)$? Think about it

Time $O(mn)$!

Randomized Algorithms: 2-dim Linear Programming

Deterministic algorithms: same behaviour on a fixed input

Randomized algorithms: random decisions

→ running time → random variable

→ output can be random

- With high probability, running time is small
- w.h.p., output is correct.

Randomized Quicksort (randomly choose pivot)

Sampling: Avoid worst cases

- With some prob, hardware can fail.
- Algorithms: repeat multiple times
  $\Rightarrow$ error probability very small

Many cases: randomized algorithms are the fastest known.
Also, simplest.

2-dim Linear Programming

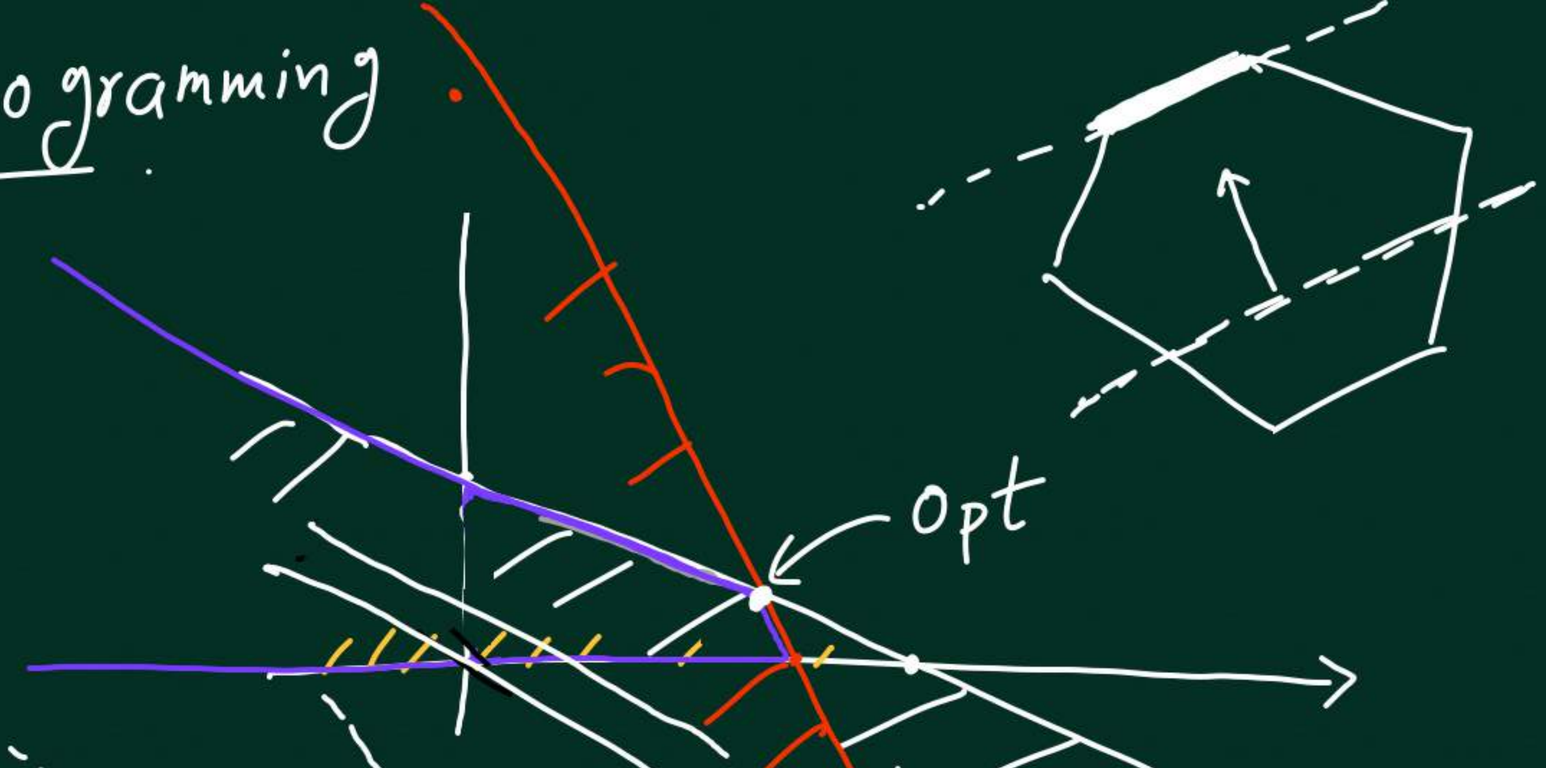
x, y

$x + 2y \le 4$

$2x + y \le 6$

$y \ge 0$

$\max\ 2x + 3y$



$(8/3, 2/3)$ Opt $\frac{22}{3}$

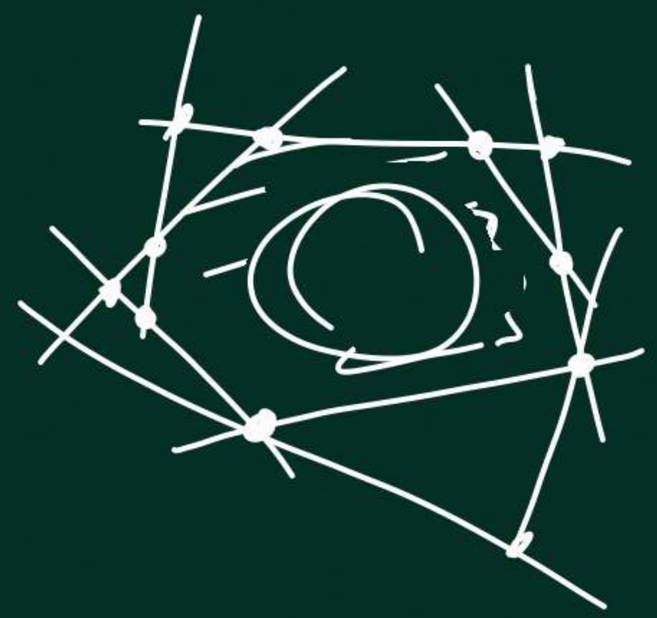
One of corners is an optimal point.

HW

n linear constraints.

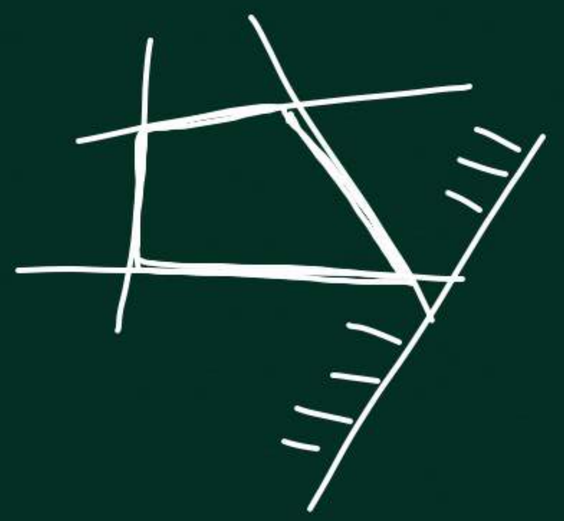
check all the corners $O(n^3)$

↳ at most n



no. of intersection points

$O(n^2)$



HW $O(n \log n)$ compute all corners.

Randomization

Find the optimal point

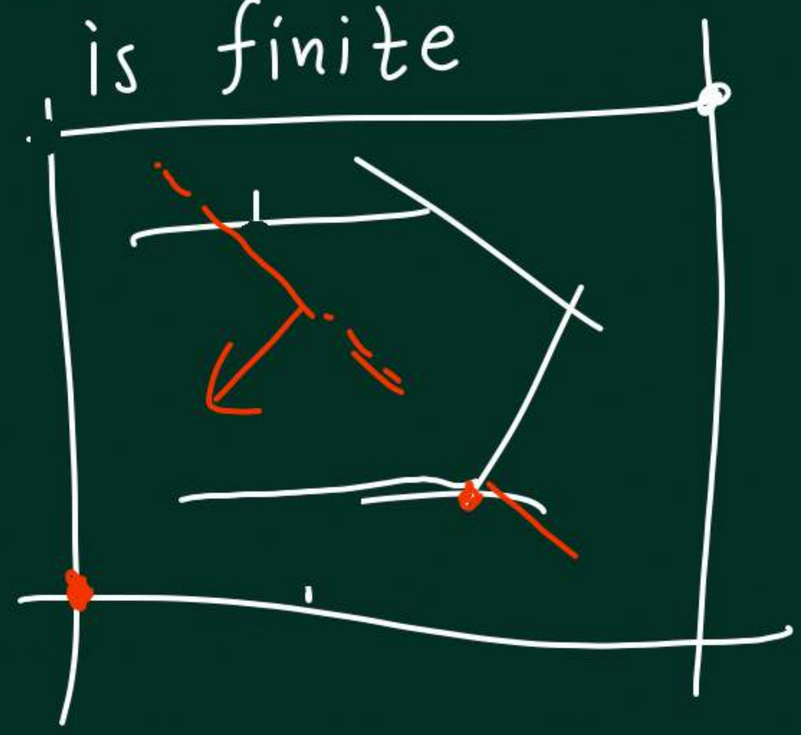
simple algorithm $O(n)$ time (w.g.p.)

Assumptions: Optimal value is finite

for every intersection point

$-L < x\text{-coord} < L$

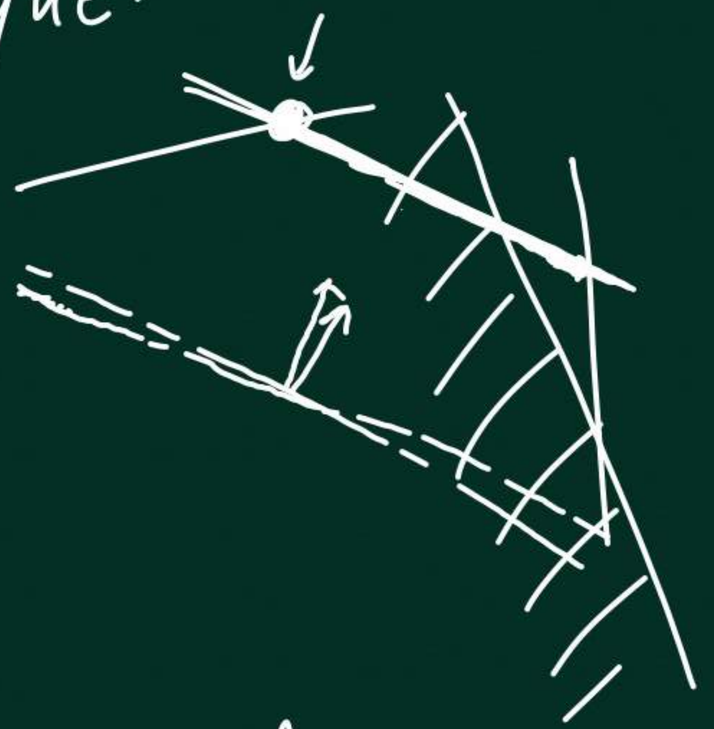
$-L < y\text{-coord} < L$



Assumption: optimal solution is unique.

$C_0 \leftarrow$ Set of constraints $\leftarrow$ $-L \le x \le L$, $-L \le y \le L$

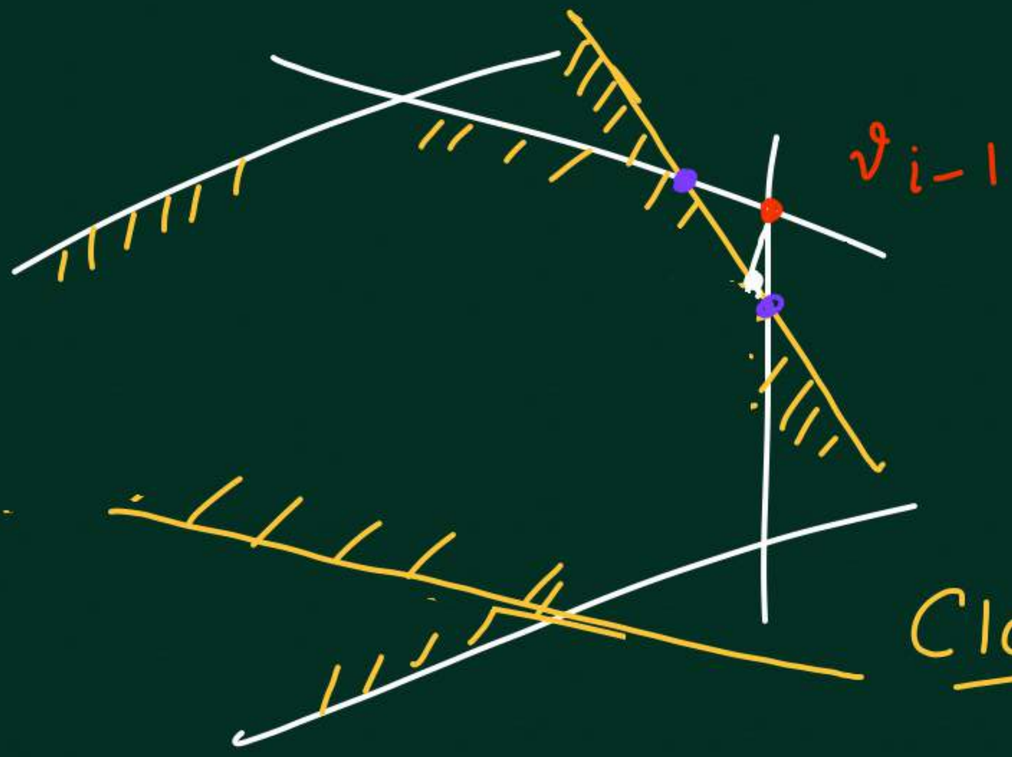
$v_0 \leftarrow$ Optimal point $\leftarrow (L, L)$



Add constraints one by one and update the optimal point

$$(C_{i-1}, v_{i-1}) \longrightarrow (C_i, v_i)$$

$$C_i = C_{i-1} \cup \{h_i\}$$



If v_{i-1} satisfies h_i then $v_i = v_{i-1}$ $O(1)$

If v_{i-1} doesn't satisfy h_i $O(i)$

Claim: v_i will satisfy hi with equality

Maximization

$f(p) = c p_x + d p_y$

↑ objective function

Obs $f(v_{i-1}) \geq f(v_i)$

Consider line segment joining v_i and v_{i-1}.

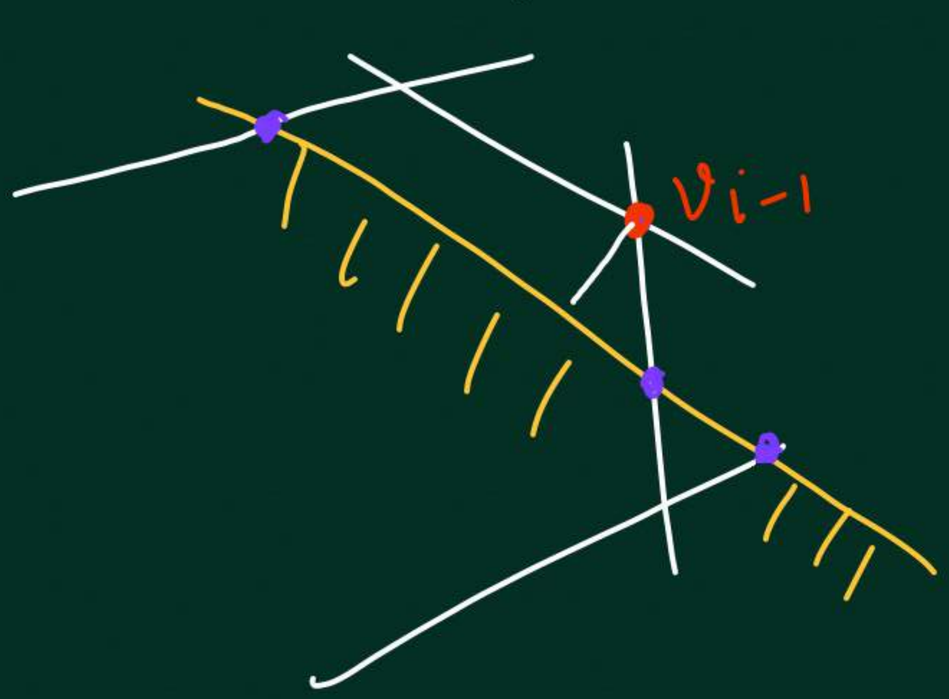
Intersection of line segment with hi is q.

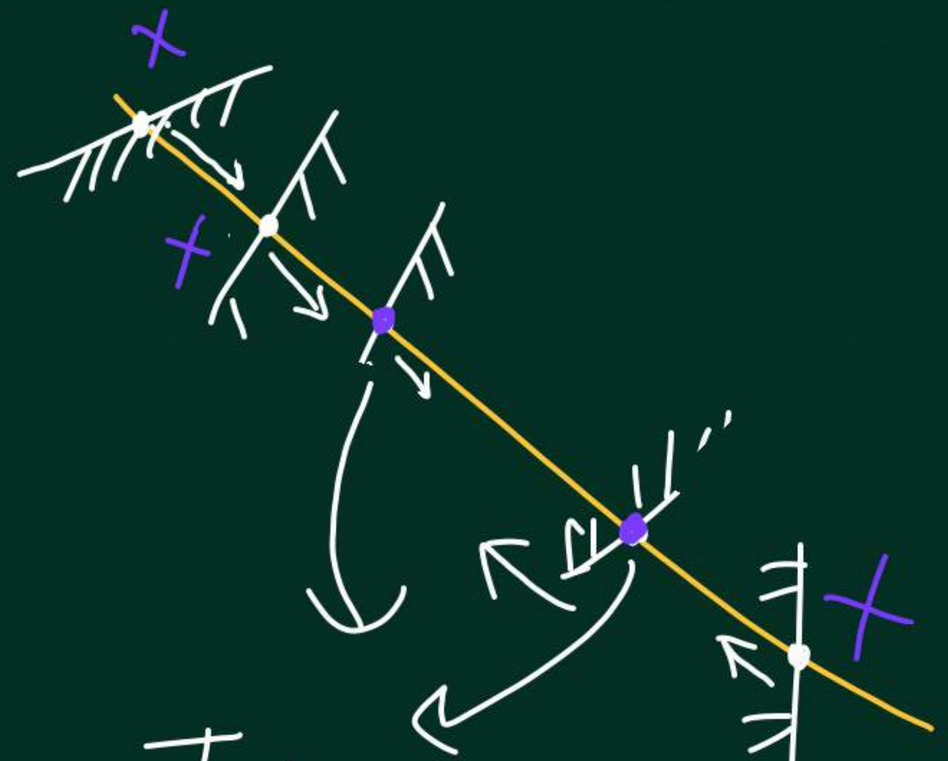
$f(v_{i-1}) \geq f(q) \geq f(v_i)$

$O(i)$ time update algorithm?

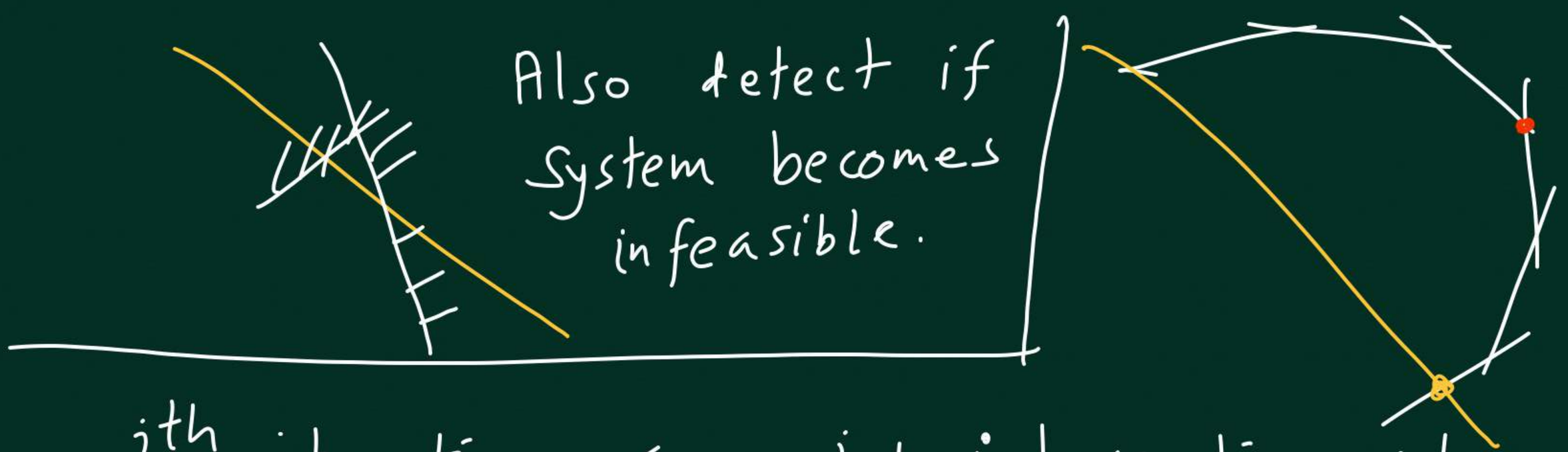
<u>Obs</u> v_i is the point of intersection of h_i and

Some h_ℓ $\quad 1 \leq \ell \leq i-1$.





Take the best of two $\leftarrow v_i$



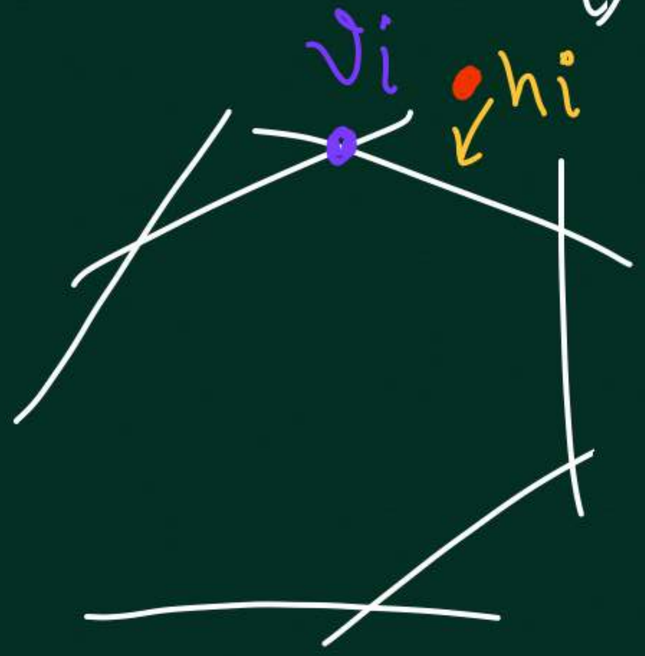
i^{th} iteration $\leftarrow$ $i-1$ intersection point

Total running time $= O(1) + O(2) + \ldots + O(i) \cdots + O(n)$

$= O(n^2)$

We arrange the lines in a random order and then run the algorithm.

Que What is the probability that $v_i \neq v_{i-1}$ (bad event)



= prob that i^{th} line is one of two lines passing through v_i

$= \frac{2}{i}$

Expected number of times i have computed intersections of lines.

X_i = the no. of intersections computed in the i^{th} iteration.

$$X_i = \begin{cases} 0 & \text{with pr } 1 - \frac{2}{i} \\ i-1 & \text{pr } 2/i \end{cases}$$

$$X = \sum_{i=1}^{n} X_i$$

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i} \frac{2}{i}(i-1)$$

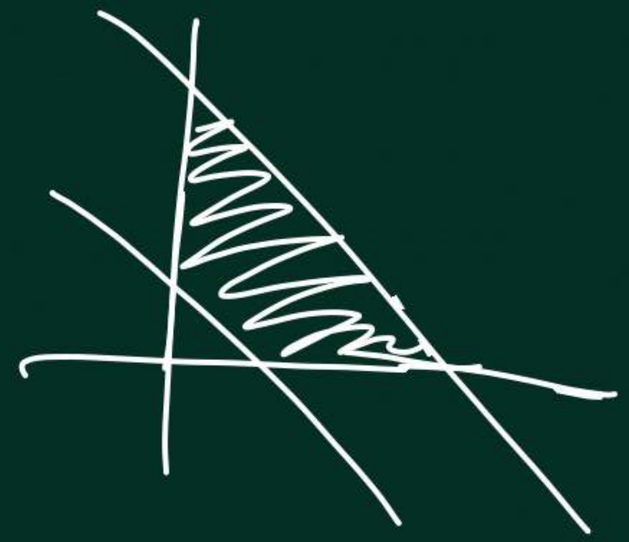
Expected running time $O(n)$

$\leq 2n$

Simplex Algorithm (not known to be in P)

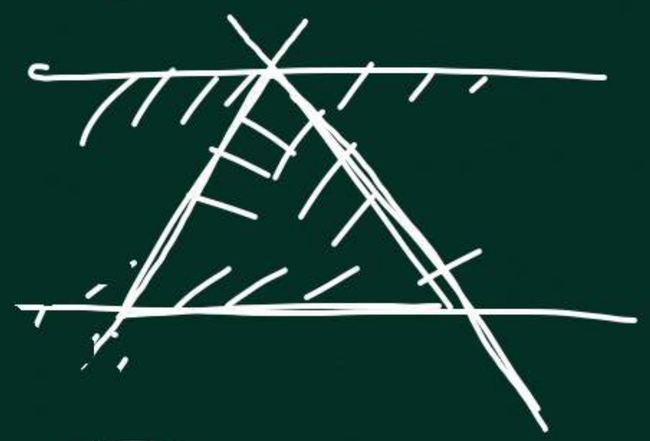
Recall: one of the corners is an optimal so

$$\begin{cases} 2x+3y \leq 5 \\ x+y \geq 1 \\ x \geq 0 \\ y \geq 0 \end{cases}$$

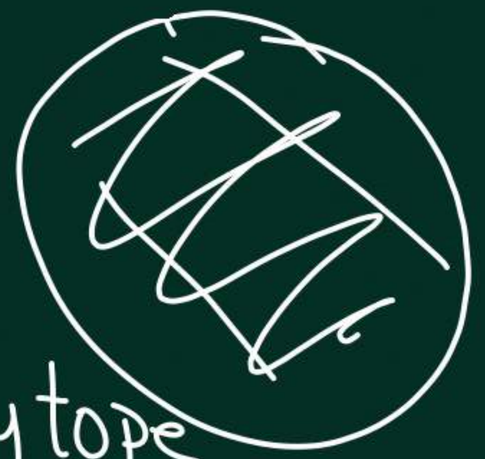


Convex set

$x+y+z \geq 1$



Polyhedron / Polytope



lution

Def (corner): For a system of linear inequalit
in n varibles, a corner is a feasible
point which satisfies n linearly
independent constraints with equality.

Def A corner is a feasible point
s.t. there exist a hyperplane H
s.t. $H \cap$ polyhedron is a unique poin

ies Simplex

- Start from a corner
- Keep going to a neighboring corner such that the objective function increases.
- Stop when there is no neighboring corner with higher objective value.

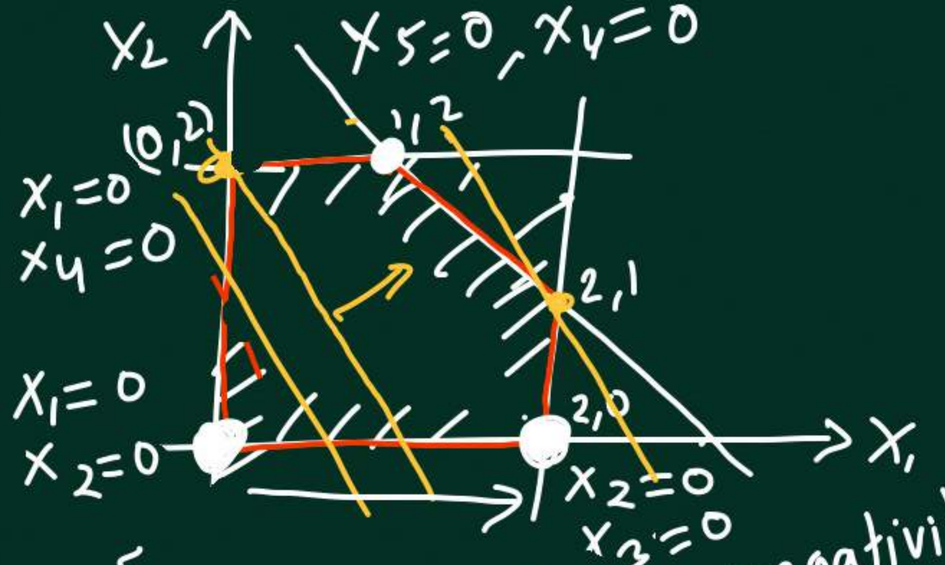
t.

n-dim hypercube

$$\left.\begin{array}{l} 0 \le x_1 \le 1 \\ 0 \le x_2 \le 1 \\ \vdots \\ 0 \le x_n \le 1 \end{array}\right\} 2^n \text{ corners}$$

Hirsch conjecture:

In n-dimensional polytope with m constraints there is always a path of length poly(n,m) between any two corners.



Initial basic feasible solution

Corner

Assume: we have an Initial BFS

$\left\{ \begin{matrix} x_1 \geq 0 \\ x_2 \geq 0 \end{matrix} \right\}$ Non-negativity Constraints

Standard Forms of linear Progra

$x_1 \geq 0$, $x_2 \geq 0$

$x_1 \leq 2$

$x_2 \leq 2$

$x_1 + x_2 \leq 3$

Max $2x_1 + x_2$

$x_1 = 2 - x_3$, $x_3 \geq 0$

$x_2 = 2 - x_4$, $x_4 \geq 0$

$x_1 + x_2 = 3 - x_5$ $x_5 \geq 0$

Start with

$x_1=0, x_2=0 \quad x_3=2 \quad x_4=2 \; x_5=3$

s. $obj = 2x_1 + x_2$

ams Allow one variable to increase

Let me choose to increase x_1

x_2 will remain zero.

$x_3 \geq 0 \Rightarrow 2-x_1 \geq 0 \Rightarrow x_1 \leq 2$

$x_5 \geqslant 0 \Rightarrow 3-x_1-x_2 \geqslant 0 \Rightarrow x_1 \leq 3$

$x_1=2 \quad x_3=0$

$x_2=0 \quad x_4=2$

$x_5=1$

Rewrite the obj function in terms of x_2, x_3

$obj = 2(2-x_3)+x_2$

$= 4-2x_3+x_2$

Should increase x_2

x_3 will remain zero

$2 - x_2 \geq 0$

$x_2 \leq 2$

$x_1 + x_2 \leq 3$

$2 - x_3 + x_2 \leq 3$

$x_2 \leq 1 + x_3$

Change $x_2 \leftarrow 1$

$x_1 = 2$ $x_4 = 1$

$x_2 = 1$ $x_5 = 0$

$x_3 = 0$

rewrite obj in terms x_3, x_5

Obj $4 - 2x_3 + \_\_ - x_5$

Obj = 4 – 2x3 + x2
= 4 – 2x3 + (3–x1 –x5)
= 7 – 2x3 – x1 –x5
= 7 – 2x3 – (2–x3) – x5
= 5 – x3 – x5

The algorithm stops here because
both x3 and x5 have negative coefficients
in the objective function
Hence, we cannot increase any of the two.
The optimal value is 5 (as x3=x5=0)