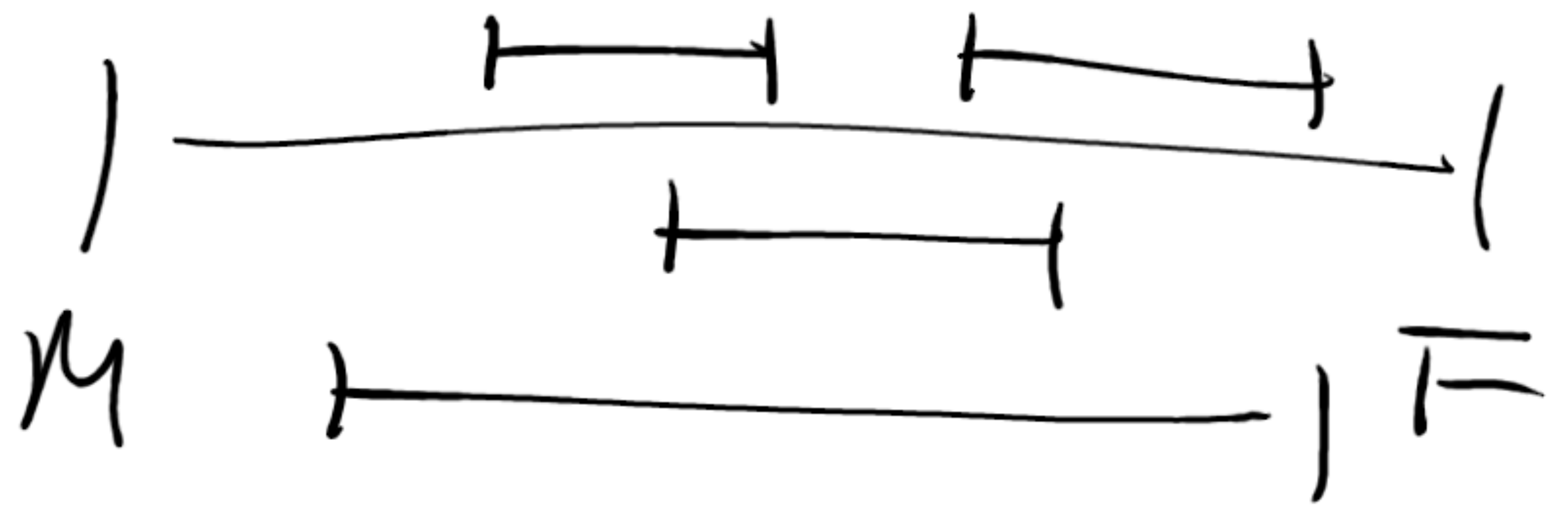
CS473 Algorithms: Lecture 3 (2022-01-25)

logistics: - pset 0 due F 1/7
- office hours: mifakes T 3:30 zoom
Pooja W 10 ——
Christian R 16 ——
- lectures online this week

last lecture: divide and conquer - closest pair

today: dynamic programming

Q: scheduling covid tests?



def: for a set of intervals $[s_1, f_1], \ldots, [s_n, f_n] \subseteq \{1, \ldots, m\}$ = $[m]$, $s_i \le f_i \; \forall i$

an interval $[s_i, f_i]$ is compatible w/ $[s_j, f_j]$ if

either - $f_i \le s_j$
- $f_j \le s_i$

A subset of intervals $S \subseteq [n]$ is feasible if $\forall i \neq j \in S$ $[s_i, f_i]$ compatible w/ $[s_j, f_j]$

convention: $\lg m \le O(\lg n)$
$\Rightarrow m \in n^{O(1)}$

Q: make money?

def: the weighted interval scheduling problem is to, given intervals $([s_i, f_i])_{i=1}^n$ and weights $v_1 \ldots v_n \in \mathbb{Z}$ compute

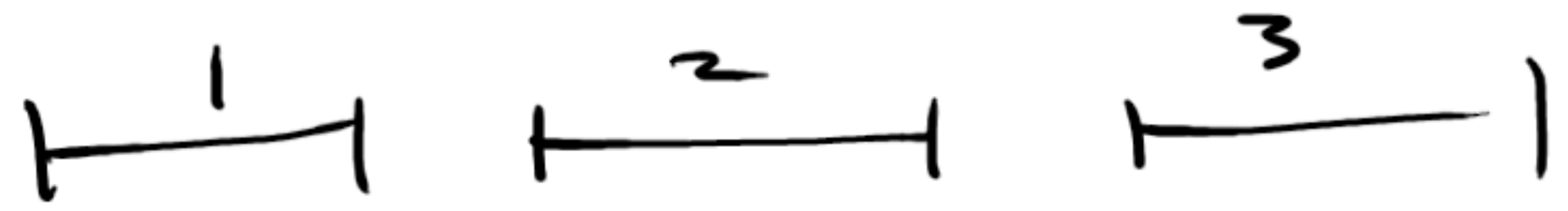
$$\max_{\substack{S \subseteq [n] \\ S \text{ feasible}}} \sum_{i \in S} v_i = OPT$$

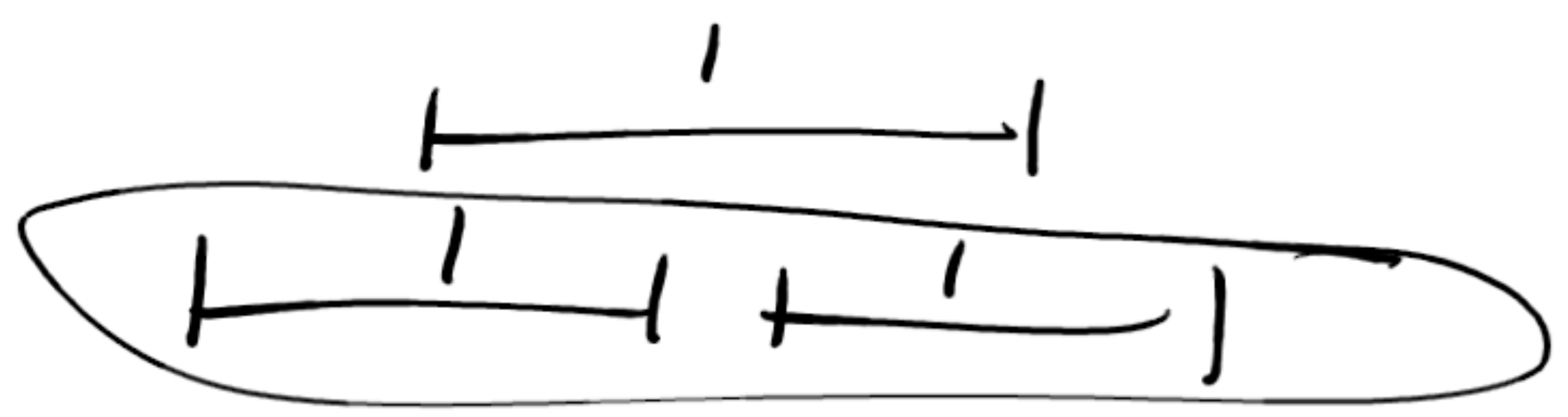
^ any

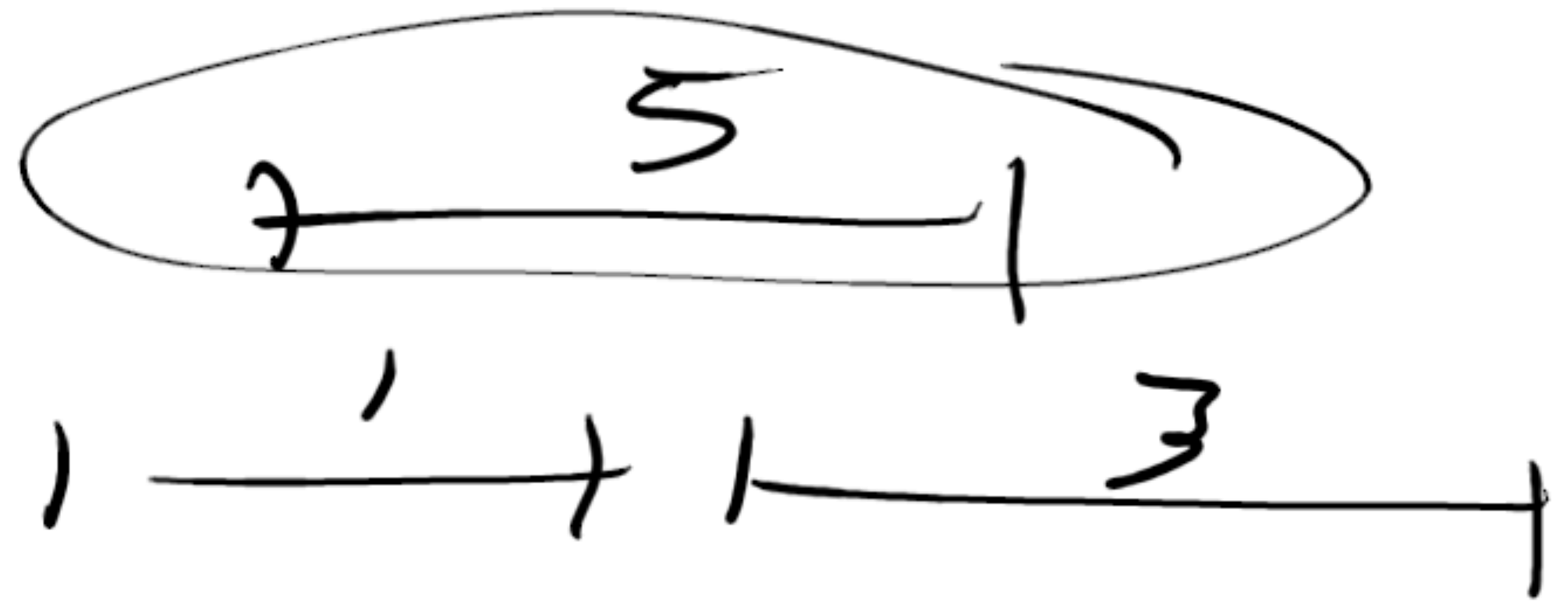
convention: $\sum_{i \in \emptyset} v_i = 0$

bm: without loss of generality all $v_i \ge 0$

Skeleton: omit interval preserves feasibility
omit negative weight intervals only can nondecrease overall value

eg:  OPT = 6

eg:  OPT = 2

eg:  OPT = 5

assumption: $f_1 \le \cdots \le f_n$

prop: weighted interval scheduling in $O(n \cdot 2^n)$ time

sketch: algo: (1) output $\max_{S \subseteq [n],\ S \text{ feasible}} \sum_{i \in S} v_i$

(2^n choices of $S \subseteq [n]$; $O(n)$ to check S feasible; $O(n)$ to compute $\sum_{i\in S} v_i$)

correctness: clear

complexity: $2^n(O(n) + O(n)) = O(n \cdot 2^n)$

Q: do better? divide and conquer?

def: for $0 \le k \le n$

$$OPT_k = \max_{\substack{S \subseteq [k] \\ S \text{ feasible}}} \sum_{i \in S} v_i$$

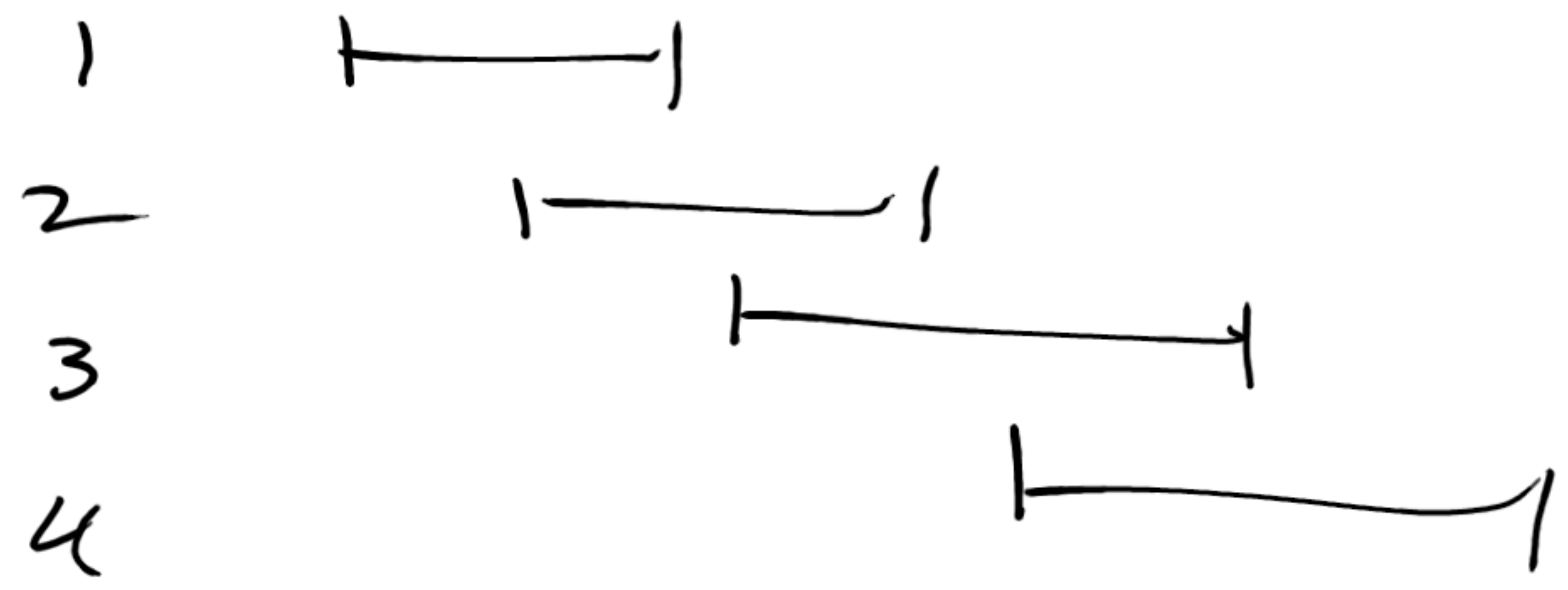
(arg: the maximizing S)

for $1 \le i \le n$ def prev(i) =

$\max \{ j : j \le i,\ [s_j, f_j] \text{ compatible w/ } [s_i, f_i] \}$

(compatible $\iff f_j \le s_i$)

convention: prev(i) = 0 if empty

eg: 

prev(1) = 0
prev(2) = 0
prev(3) = 1
prev(4) = 2

fact: prev(1), ..., prev(n) can be computed in $O(n \log n)$ time given sorted order

prop: $S \subseteq [n]$ feasible iff

either (a) $S = T \subseteq [n-1]$, T feasible

(b) $S = \{n\} \cup T$, $T \subseteq [\text{prev}(n)]$, T feasible

pf. $\Leftarrow$:

(a) $S = T \subseteq [n-1]$ feasible, clear

(b) $T = \{i_1, \ldots, i_k\} \subseteq [\text{prev}(n)]$

w/ $f_{i_1} \le \cdots \le f_{i_k} \le s_n$

$\Rightarrow$ all $j \in T$ have $[s_j, f_j]$ compatible w/ $[s_n, f_n]$

T feasible $\Rightarrow$ $S = \{n\} \cup T$ feasible

$\Rightarrow$: (a) if $S \subseteq [n-1]$, S feasible $\Rightarrow$ $S = T \subseteq [n-1]$ feasible

(b) if $S \not\subseteq [n-1]$, then $S = \{n\} \cup T$

$T \subseteq [n-1]$

$T = i_1 \le \cdots \le i_k$

w/ $f_{i_1} \le \cdots \le f_{i_k}$

all $j \in T$ $[s_j, f_j]$ compat w/ $[s_n, f_n]$

$f_n \ge f_j$ by sorted-ness

$\hookrightarrow$ $f_j \le s_n \Rightarrow j \le \text{prev}(n)$

$\Rightarrow T \subseteq [\text{prev}(n)]$ ∎

rmk: decomposing feasibility on $[n]$ into two (disjoint) subsets

(disjoint — not strictly needed)

cor: $OPT_n = \max\{OPT_{n-1}, OPT_{\text{prev}(n)} + v_n\}$

pf. $\hookrightarrow$ $\max\limits_{\substack{S \subseteq [n] \\ S \text{ feasible}}} \sum_{i \in S} v_i = \max\left\{ \max\limits_{\substack{S \subseteq [n-1] \\ S \text{ feasible}}} \sum_{i \in S} v_i, \right.$

$$\left(\max_{\substack{T \subseteq [\text{prev}(n)] \\ T \text{ feasible}}} \sum_{i \in T} v_i \right) + v_n = \left[\max_{\substack{S = \{n\} \cup T \\ T \subseteq [\text{prev}(n)] \\ T \text{ feasible}}} \sum_{i \in S} v_i \right\} = \left(\sum_{i \in T} v_i\right) + v_n$$ ∎

also: solve(k) = (1) if $k = 0$, return 0

(2) return $\max\{\text{solve}(k-1), \text{solve}(\text{prev}(k)) + v_k\}$

prop = solve(n) computes OPT_n in $O(2^n)$ time

pf = correctness: clear

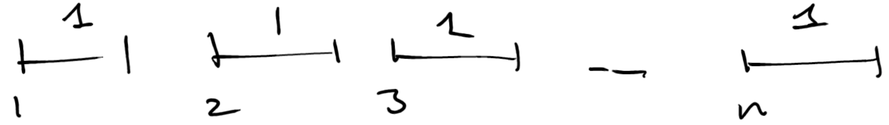
complexity: $T(k) = \max_{j \le k}\{\text{running of solve}(j)\}$

$T(k) \le T(k-1) + \underbrace{T(\text{prev}(k))}_{\le T(k-1)} + O(1)$

$= 2T(k-1) + O(1) = O(2^n)$ ✓

rmk = better than $O(n2^n)$, but not by much

Q: can we do better?

eg = 

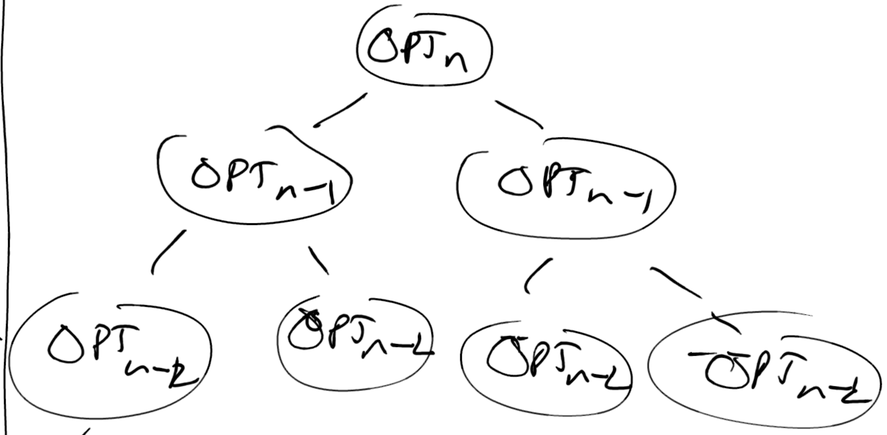
$\Rightarrow \text{prev}(k) = k-1 \ \forall k$

Q = solve(n) takes $\Omega(2^n)$ time

skeptic: $T(k) = T(k-1) + T(k-1) + \Theta(1) \ge \Omega(2^n)$ ☹

rmk = counterexample is tricky, no trivial fix

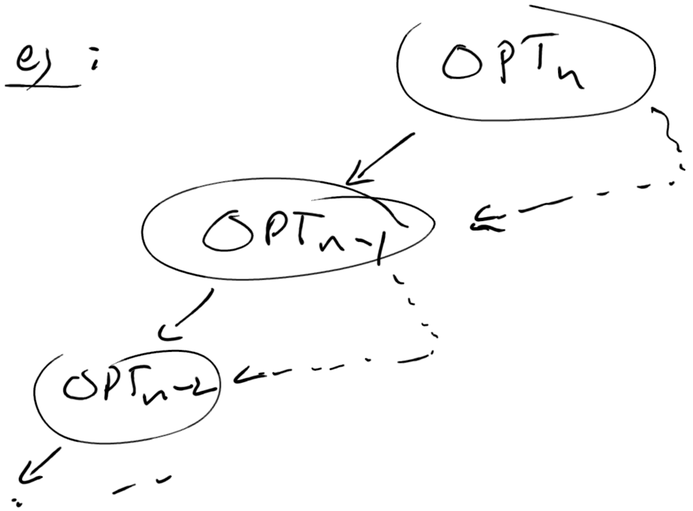
Q: can we do better?



idea (dynamic programming): solve each subproblem [at most once], by [storing] the solution

↳ memoization

eg:



also - global array $M[\cdot]$ on $[1, \ldots, n]$

solve-DP(k) = (1) if $k=0$, return 0

(2) if $M[k]$ empty

$M[k] = \max\{$ solve DP$(k-1)$, solve DP(prev[k]) $+ v_k\}$

(3) return $M[k]$.

prop: solve DP computes OPT_n in $O(n)$ time

pf: correctness: clear

complexity: clm: runtime is $O(1) \cdot$ (# recursive calls)

clm: # recursive calls is $\leq 2n$

pf: subclm: throughout the algorithm, the following invariant holds

$2($# empty cells in $M) +$ (# recursive calls) $= 2k$

pf: by induction on algo

initialization: $2(n) + 0 = 2n$

within algo:

	Δ # empty cells	Δ # recursive calls
$k=0$:	0	0
$M[k]$ non empty:	0	0
$M[k]$ empty:	-1	2

$\square$

Q: find the [solution]?

prop: $S \subseteq [n]$ feasible iff

(a) $S = T \subseteq [n-1]$ feasible

(b) $S = \{n\} \cup T$, $T \subseteq [prev(n)]$, T feasible

cor: exists optimal soln $S_0 \ni n$ iff

$OPT_{prev(n)} + v_n \geq OPT_{n-1}$

rat: $OPT_n = \max(OPT_{n-1}, OPT_{prev(n)} + v_n)$

type (a) type (b)

iff type (b) solution achieve opt value

iff type (b) soln value $\geq$ type (a)

iff

algo: global array M[·], N[·]

soln-DP(k) = (1) if k=0, return $(\emptyset, 0)$

index → ← index 1, ← value

(2) if M[K] empty

(a) if soln-DP(prev(k))[1] + v_k $\geq$ soln-DP(k-1)[1]

M[K] =

$O(n)$ N[K] = {k} ∪ soln-DP(prev(k))[0]

actual soln

(b) else M[K] =

N[k] = soln-DP[k-1][0]

(3) return (N[K], M[K])

prop: soln-DP finds opt [soln] in $O(n^2)$ time

sketch: correctness: clear

complexity: clm - # recursive calls $\in O(n)$

clm: time $\in O(n) \cdot$ (# recursive calls)

□

Q: do better?

Also: global array M[·], init by solve-DP(n)

soln-DP-fast(k)

(1) if k=0 output 0

(2) if M[prev(k)] + v_k ≥ M[k-1]

(a) output k

(b) recurse soln-DP-fast(prev(k))

(3) else recurse on soln-DP-fast(k-1)

Prop: soln-DP-fast finds opt soln in O(n) time

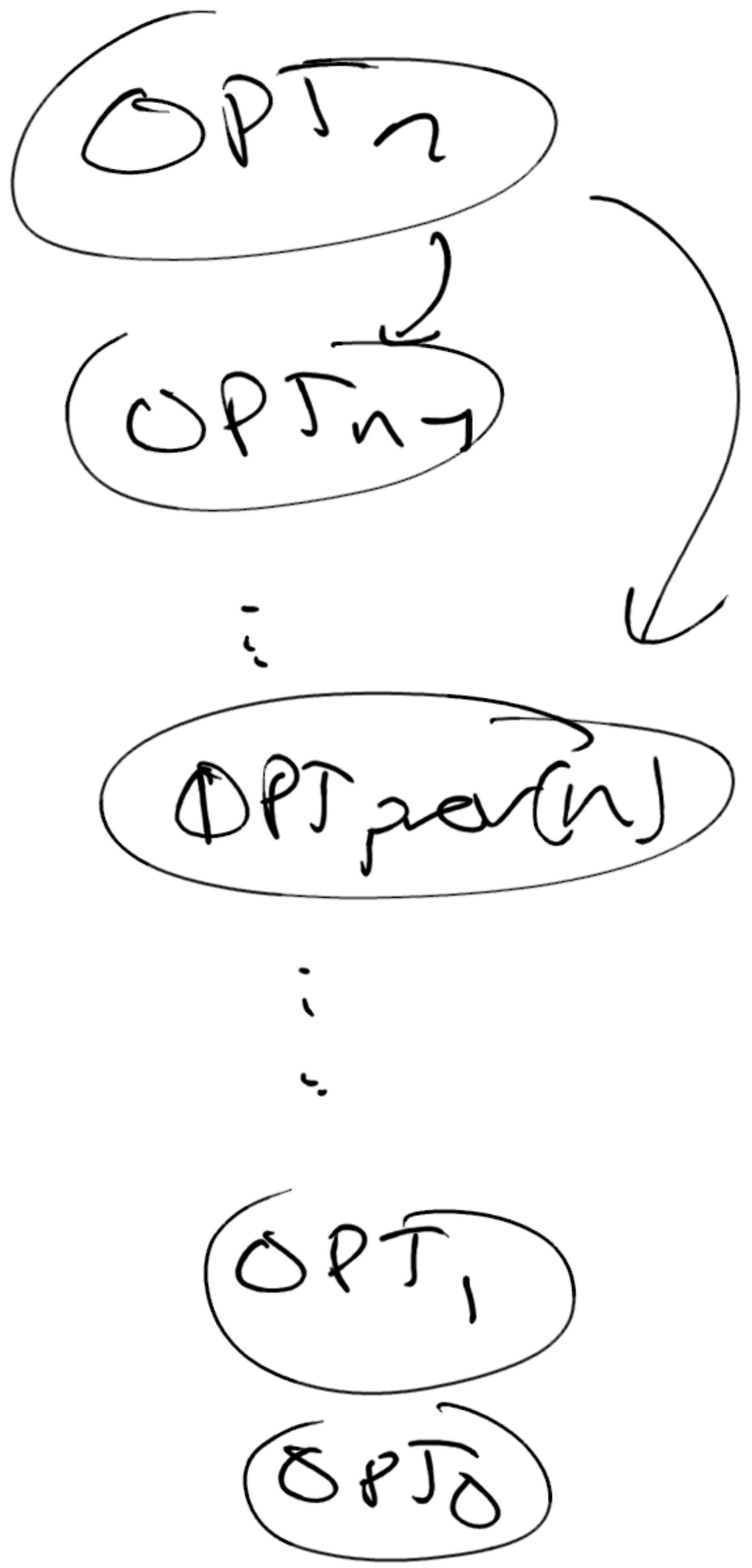
Pf: correctness: clear

complexity: O(n) + T(k)

$\leq \max\{T(prev(k)), T(k-1)\} + O(1)$

$\leq O(k)$ ∎

Q: non-recursive algo?



algo: solve-iter(n):

(1) $M[0] = 0$

(2) for $k = 1, \ldots, n$

(a) $M[k] = \max\{M[k-1], M[prev(k)] + v_k\}$

(3) output $M[n]$

prop: solve-iter computes OPT_n in $O(n)$ time

pf: correctness: clear

↳ obtained when algo reaches this point

complexity: $O(n)$

this course: [strongly] prefer iterative algo

today: dynamic programming

weighted interval scheduling

- brute force $O(n 2^n)$
- recursive $O(2^n)$
- memoized recursion $O(n)$
- iterative $O(n)$

logistics: pset 0 due F 17

office hours: [illegible] T 3:30 zoom

Poor ~ 10 ———

Christa R 16 ———

lectures all this week